UTRECHT UNIVERSITY

ALGORITHMIC DATA ANALYSIS DEPARTMENT

MASTER THESIS

Influential ML: Towards detection of algorithmic influence drift through causal analysis

Author:

Sergey Dragomiretskiy (6981003)

University Supervisor:

prof. habil. Georg Krempl

prof. dr. A.P.J.M. Siebes

FACULTY OF SCIENCE

MSC BUSINESS INFORMATICS

November 25, 2021

Abstract

Machine Learning (ML) algorithms are applied in environments where they can make increasingly more impactful decisions. Such decisions can have great power over the classified instances. Therefore, these classifications need to be accurate, reliable, and explainable. Pushed mainly by regulations, a new ML field for research and practice is gaining traction: MLOps. This consists of reliably and efficiently deploying and maintaining ML models in production. Influential ML is part of MLOps, which focuses on understanding the consequences of an ML algorithm. One of these consequences is prediction influence drift, whereby the classifications of an algorithm are the cause for changing instances over time through feedback loop effects. These feedback loops can be self-fulfilling or self-defeating prophecies. These feedback loops are researched, and a categorization is created based on the literature review.

This research aims to create a detection approach to identify, quantify, and possibly localize prediction influence drift. First, a synthetic streaming dataset is created, which contains a self-fulfilling or self-defeating feedback loop. Secondly, a classifier is created, which classifies in a trivial checkerboard prediction pattern with a certain % of instances flipped to create interventions on observational data. Based on this experiment setup, a prediction drift detector is created with the goal to classify, quantify, and localize this prediction influence drift. The result is a developed detection approach that uses causal inference techniques with distribution KDE change methods to detect self-fulfilling and self-defeating prophecies on simulated synthetic data. The evaluation of this approach concludes that this detector is sufficiently good at detecting drift on various parameter settings. Thus, the method looks promising, however, it still needs some additional complexity to be used in a real-world dataset.

Contents

Chapter 1

Introduction

1.1 Problem statement

Machine Learning (ML) is becoming an important technique to solve data-driven problems. Companies are productionizing ML models more often with varying results. Productionizing ML models is a complex task and comes with many new challenges. While big companies which invested early in these solutions managed to launch impactful large-scale systematic prediction systems using ML, other companies are still struggling with the implementations and are failing to get their projects through the proof-of-concept phase.

ML algorithms only add value when put online into production, where users can interact with them. However, it is not realistic to train an ML model once and expect it to perform well continuously. Possible underperformance can be caused by a changing world where the algorithm should keep up with the change. One way for the model to keep up with the changing environment is by gathering new data and retraining the model. However, this option is not viable in the long term. It leads to high maintenance costs if changes are frequent because new data needs to be gathered and the model retrained frequently. Another way is to use online learning. Online learning is when algorithms in production continuously learn using incoming data from the environment they are in. This learning makes online learning systems dynamic and able perform well over time. Being able to change is a preferable feature for an ML system. However, this brings with it new challenges that are yet unsolved.

One of the challenges is realising what the consequences will be of putting a self-learning prediction system in a specific environment. A possible consequence of introducing such a system to an environment is that the model starts influencing the environment that it is in. Through learning, that environment can influence the model, thereby creating a feedback loop by introducing and applying the algorithm. Here the algorithm is causing the environment to change in unprecedented and possibly unwanted ways. This effect is called prediction drift, and it is part of a larger area called influential ML. This drift can happen directly or indirectly and can be desirable or adversarial.

Systematic bias through feedback is not new. Any systematic process can have these kinds of feedback loops. An example of this is in the social influential context, where a behavioural feedback loop in popularity of items exists because society relies on aggregated opinions of others to make decisions. However, putting ML in production can amplify feedback loop effects through self-learning if the learning is not monitored or constrained. Without considering prediction drift beforehand or noticing it in production, the system might deviate into an undesired state, with possible side effects being an inferior performance of the model.

Prediction drift needs to be detected and understood. However, there are no proven ways of detecting prediction drift. This research will focus on understanding different feedback loops, proving these effects in an ML context, and creating a detection mechanism to detect when such loops cause drift.

1.2 Research Objectives

This thesis builds upon the research done by Jelsma et al. (2021), where current drift methods are analysed and intrinsic drift is explained. The aims of this research are to build further upon the topic by creating an overview of feedback loops that could arise from ML prediction drift, researching methods that could detect such drift, and creating and validating a prediction drift detection method.

1.2.1 Research Questions

RQ1: *What kind of feedback loops are created by systematic classification systems?*

First, the field of systematic classification systems will be researched to find areas and tasks where systematic classification systems are active. This analysis is done to create a clear picture of the impact of the problem and to define the technicalities and the subcategorisation of the different kinds of feedback loops. One specific feedback loop is chosen for the following research questions.

RQ2: *What is the formal generation process of the data behind the feedback loop chosen in RQ1?*

After selecting a specific feedback loop from the results of RQ1, domain data is gathered, which can have this feedback loop in it.

SRQ1: *Which data can be used for the specific feedback loop detection?*

A data source has to be found and checked whether it can contain a feedback loop. After a positive suitability check, a formal data generation process based on that dataset needs to be defined to create new data. If such data source can not be found, then this data needs to be synthetically generated.

RQ3: *What kind of prediction drift detectors are currently available?*

Current prediction drift detectors are examined to get an overview of the research space. Two points are essential when looking at the prediction drift detectors.

SRQ1: *For what feedback loop mechanisms are the different prediction drift detectors suitable?*

The found detection mechanisms will be paired with the feedback loops found in RQ1. The feedback loops that do have detection mechanisms can be proposed to move into the detection application domain. The feedback loops that do not have a detection mechanism can be proposed for further foundational research where better techniques are needed.

SRQ2: *What are the assumptions and limitations of the current methods?*

Each detection mechanism has its own downsides. These downsides are found and used for further improvement in this research.

SRQ3: *How is causal inference linked with prediction drift?*

Prediction drift is when the introduction of a model changes the environment. This drift implies a causal structure where the model is causing the change. Research should be done on how this is formalised.

RQ4: *How can prediction drift be detected in the feedback loop chosen in RQ1 by using causal inference?*

Using the knowledge from RQ3 and the data generator from RQ2, a specific prediction drift detection approach can be created for the feedback loop chosen in RQ1.

SRQ1: *If it is possible, how can a prediction influence detector be made to achieve this?*

The goal of this research will be to create a detection approach and validate it on the generated data from RQ2.

1.3 Research Method

In the book "Design Science Methodology", Wieringa (2014) introduced the engineering cycle, which is a rational problem-solving process. In this cycle, the validated treatment is transferred to the real world, used, and evaluated. This validated treatment is the outcome of the design cycle. However, if transferring to the real world will not be possible due to resource or time restrictions, only the design cycle is followed, which skips the treatment implementation phase. The engineering cycle consists of the following phases: the problem investigation, the treatment design, the treatment validation, and the treatment implementation. Each phase is elaborated on for this research: The engineering cycle

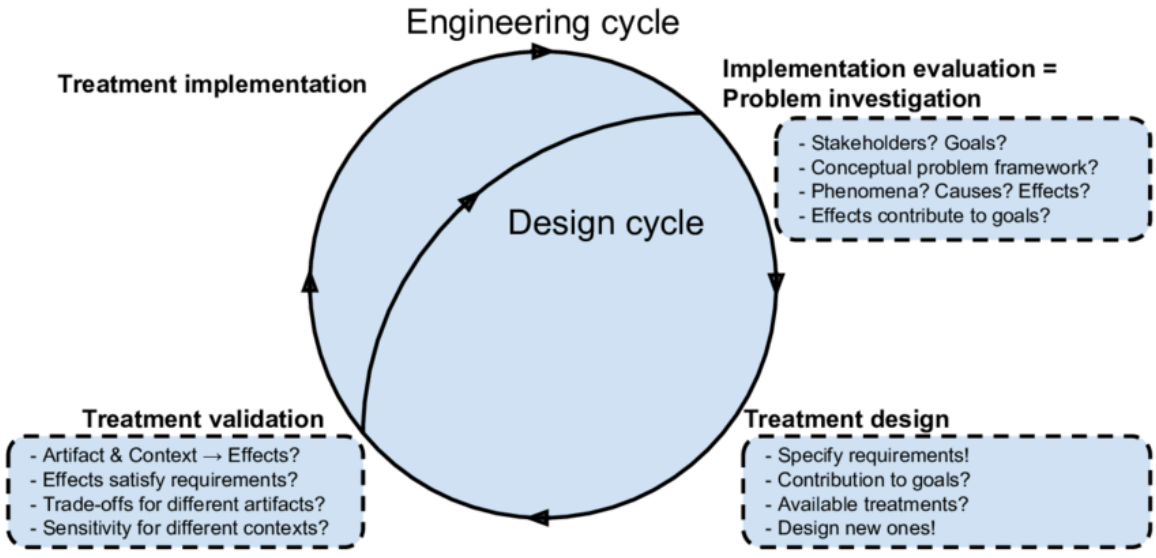


Figure 1.1: *Engineering Cycle Wieringa (2014)*

consists out of the following phases and can be seen in Figure 1.1:

1. Problem investigation:

In this phase, literature is reviewed, and research gaps identified to find what must be improved and why. This study mainly draws on six streams of research: productionizing ML, varieties of drift, drift detection, feedback loops, prediction drift, and the causal nature of prediction drift.

2. Treatment design:

A data generator, a detection approach, and an evaluation method are designed on a selected feedback loop from the problem investigation.

3. Treatment validation:

The detection approach can be validated using the data generator. If this validation is satisfactory, continuation to treatment implementation is the next step. If not, the next step will be to go back to the problem investigation step to determine what caused the unsatisfactory results.

4. Treatment implementation:

The created detection approach can be generalized and standardized by applying it to other datasets to check for prediction drift.

1.4 Literature Research Protocol

Given the research questions defined above and design cycle elaboration, a literature research protocol can be proposed. First, papers are searched for that are needed for the problem investigation describing the details of the concepts used in this research. Secondly, a search for papers is done where feedback loops exist in the real world. Then, a search is done for the prediction drift detection mechanisms. Finally, research is found that describes the limitations and the evaluation techniques of detection mechanisms.

Most topics mentioned in the problem investigation phase in the engineering cycle have extensive survey papers from where the literature study can start. For example, for concept drift, there is a survey paper from Žliobaitė et al. (2016) , which explains different concept drifts and the applications where these are in. Starting from these survey papers and snowballing forward from the cited papers will provide the foundation for this research.

1.5 Implementation

For the analysis of the datasets, the implementation of the data generator, and the creation and evaluation of drift detection approaches, Python 3 is used as the programming language. This choice is made because Python is currently the most popular language for coding ML projects. The code will be shared to a Github repository and will be added to open-source libraries after validation. River is the main package that the implementation of this research will be based on (Montiel et al., 2021). It is a machine learning library for dynamic data streams and continual learning. It also provides state-of-the-art data generators and detection mechanisms. The evaluation of the detection drift mechanisms will be done by methods that arise from the literature review. Standard python packages that also will be used are scikit-learn, numPy, sciPy, and matplotlib.

Chapter 2

Related Literature

2.1 Machine Learning

Machine learning (ML) is a branch of artificial intelligence that systematically applies algorithms to synthesize the underlying relationships among data and information (Alpaydın, 2010). This research will focus mainly on classification ML tasks. Thus, these will be assumed when talking about ML further on. The field of ML is increasingly growing in the amount of research being published and the applications being developed. ML is applied more consistently and in more impactful applications. The end goal of ML is primarily automated decision making. Specific actions can be chosen for that instance at a given moment by classifying specific outcomes given input data. For this automated decision making to be useful, a machine learning algorithm first needs to be trained and deployed online. The usefulness of ML solutions is still not entirely accepted and there is still hesitation to use these for real-world use-cases. Trust is of significant importance in the acceptance of new technologies (Söllner et al., 2016). For long term trust, it is needed to know that the ML system will perform well, no matter the circumstances. However, ML system performance is known to deteriorate over time (Cerquitelli et al., 2019). This performance needs to be constantly monitored, and specific actions need to be suggested and performed when lower performance triggers. This monitoring is part of a relatively new practice and an active research area called MLOps (Atwal, 2020).

2.2 MLOps

MLOps is a ML version of DevOps where the development of traditional software and its operations in production are combined in one process (Bass et al., 2015). Currently, the definition of such a method is still emerging. Some research has already been done on this, and many names have been proposed for similar methods already: DataOps (Capizzi et al., 2020), AIOps (Mohanty and Vyas, 2018), MLOps (Fursin et al., 2020), and ModelOps (Hummer et al., 2019). MLOps will be used to name such a method in this research because this term was most common in related literature. The goal of MLOps is to provide a method to the growing field of ML so that teams can go from ad-hoc ways of addressing problems in the software lifecycle to mature problem analysis and ML model building, training, deploying, and evaluating. The value of MLOps can be significant by ensuring high service quality, customer satisfaction, boosting engineering productivity, and reducing operational cost (Dang et al., 2019). A classical ML process is CRISP-DM which describes the ML lifecycle in six phases: business understanding, data understanding, data preparation, modelling, evaluation, and deployment (Chapman et al., 2000). This model assumes most of the phases happen offline. However, it is not feasible to repeatedly do these phases manually in a changing environment. An automated MLOps process is needed for a learning system to stay consistent over time. The evaluation phase of ML models is the principal part to focus on when a model needs to fit a dynamic environment since this is where the monitoring is done. In a static environment, data can be collected once, the model trained after that and deployed on production when test metrics are satisfactory. However, problems are introduced when data needs to be collected over time. The main problem is that data from a later time can be different from the data the model was trained on since the underlying data is likely to change over time. This change violates the fundamental assumption for static ML: the training and test data are drawn from the same distribution. Drift detection on algorithms that learn online is a safeguard for this assumption. This detection provides evidence that something about training is not consistent when testing. Online learning algorithms can also provide the solution to the requirement that algorithms should change based on the changing environment.

2.3 Online Learning

For machines to learn on the go, they need to apply a particular form of continuous learning. Online learning algorithms process each training instance once and maintain a current hypothesis that reflects all the training instances seen so far (Oza and Russell, 2001). This kind of learning is needed in a dynamic environment because hold-out test sets may be incomplete or reflect biases inherent to the system it is collected from. For example, in a credit lending context with a default prediction task, only accepted loans data are collected and not on rejected loans. Another argument by Sculley et al. (2015) states that creating static ML systems accumulates technical debt later. Applying unit tests to individual components and end-to-end tests will not stay valid in a changing world. Because of entanglement, changing the input distribution of an algorithm may change everything: the features, weights, and importance. The proposed solution for this is detecting changes in prediction behaviour as they occur. For the implementation of online learning with online detection approaches, common datasets are not suitable.

2.4 Data streams

Data streams are the most suitable for ML systems applying online learning where detection is needed. Individual data items may be relational tuples in a data stream, and their continuous arrival is in a multiple, rapid, time-varying, possibly unpredictable and unbounded stream (Babcock et al., 2002). These differ from conventional stored relation models in several ways: the data elements arrive online, the system has no control over the arrived order, data streams are potentially unbounded in size, and once an element is processed, it is discarded or archived. Mining big data streams faces three principal challenges: volume, velocity, and volatility (Krempl et al., 2014).

For the sake of this research, volatility is the main focus. Volatility corresponds to a dynamic environment with ever-changing patterns. Old data is of limited use in this case, even if it could be saved and processed again later. This limited use is due to changes that can affect the induced data mining models in multiple ways: change of the target variable, change in the available feature information, and drift. For the credit lending prediction example, volatility will be seen in the changing target variable of comparable cases when the definition of default vs non-default changes due to business or regulatory requirements. Read (2018) showed that concept-drifting data streams can be treated

as time series. This finding is useful for this research since the finance domain primarily uses time series data. Compared to data streams, time series data instances are ordered by date, whereas in data streams, time might not even be a used feature.

ML algorithms that apply online learning using data streams automatically learn from newly incoming data. Because of this, it is essential to evaluate whether the algorithm is learning correctly quickly. Inadequate learning can happen when the environment starts changing and drifts the model because of that. The process of finding this inadequate learning because of drift is called online detection.

2.5 Drifts and drift detection

Drift in data is a long old problem that comes hand in hand with designing learning systems in complex environments. Researchers were already tackling this problem in the starting years of machine learning by making models that determined drift (Schlimmer and Granger, 1986). There are different kinds of drifts that can negatively impact the performance of a model. The general term for the deterioration of a model is called model drift. Model drift can be split into specific kinds of drifts. Intrinsic drifts are fundamentally present in ML algorithms because of the environment it is in. Intrinsic drifts include data drift, concept drift, and dual drift (Quiñonero-Candela et al., 2009; Moreno-Torres et al., 2012; Gama et al., 2014). Dual drift is a combination of data drift and concept drift. A special kind of drift is prediction drift. In contrast to intrinsic drifts, prediction drift is caused by the algorithm itself. The elaborations on the different kinds of drifts are given below.

Formally, a model can be learned given some data and a modelling framework. This framework is defined using the following notations (Moreno-Torres et al., 2012):

- A set of features or covariates x
- A target variable y (the class variable)
- A joint distribution $P(y, x)$

The model can be used to make predictions $P(y \mid x)$ for some targets y given some new covariates x. Drifts appear when training and test joint distributions are different.

$$Drift : P_{tr}(y, x) \neq P_{tst}(y, x)$$

.

2.5.1 Data drift

With other names being feature drift and covariate shift, data drift occurs when there are differing training and test data distributions. This drift appears when $P_{tr}(y \mid x) = P_{tst}(y \mid x)$ and $P_{tr}(x) \neq P_{tst}(x)$. This drift can happen when previous infrequent or unseen feature values become more frequent, and the relationship between the feature and the target stays the same. For the sake of this research, this drift is related to the prior probability drift where $P_{tr}(x \mid y) = P_{tst}(x \mid y)$ and $P_{tr}(y, x) \neq P_{tst}(y, x)$. In the prior probability case, the class balance is different from training and test.

Moreno-Torres et al. (2012) consider two leading causes of data drift: The first cause of data drift is sample selection bias. This bias is when the distributions differ due to an unknown sample rejection process (Quiñonero-Candela et al., 2009). Selection bias arises from the data generating process or from the data cleaning step where specific data is rejected or not collected at all. The second cause of data drift can be a non-stationarity environment. Examples of this can be seen in ML tasks that are dependent on time, like forecasting. The incoming data from one period might be of a different distribution than another period.

Real-life reasons for this non-stationary environment can involve changes in the domain where the feature measurements can change or that the incoming data is from a vendor which changes its business logic rules.

Data drift detection: The input data to the model needs to be monitored and compared to a holdout dataset to detect data drift. This comparison checks whether the statistical properties of both datasets are significantly different. This checking can be done using a variety of methods. Examples of these are the Population Stability Index, the Kolmogorov Smirnov statistic, Histogram intersection metrics, or Z-score/T-score statistics.

2.5.2 Concept drift

Concept drift describes the phenomenon of a changing class definition over time in machine learning for data streams (Widmer and Kubat, 1996). This changing definition can also come from changing the statistical properties of the target. If in a ML setting, the data generating source of the training

data differs from the data generating source of the testing data, then it is said to have a concept drift (Žliobaitė, 2010). Gama et al. (2014) provide a formal definition for a concept.

$$Concept : P_t(X, Y) \; at \; particular \; time \; t$$

Concept drift occurs between times t and u when the distributions change,

$$Concept \; drift : P_t(X, Y) \neq P_u(X, Y)$$

The core assumption when dealing with the concept drift problem is uncertainty about the future. Concept drift can be divided into real and virtual concept drift (Oliveira et al., 2021), be quantified in magnitude, duration, and path length (Brzeziński and Stefanowski, 2011), and can have four forms: sudden drift, incremental drift, gradual drift, and recurring drift (Figure 2.1) (Žliobaitė, 2010). Changes in underlying data occur due to changing personal interests, changes in population, adversary activities, or the complex nature of the environment (Žliobaitė, 2010).

Concept drift detection: Detecting concept drift is more challenging compared to other intrinsic

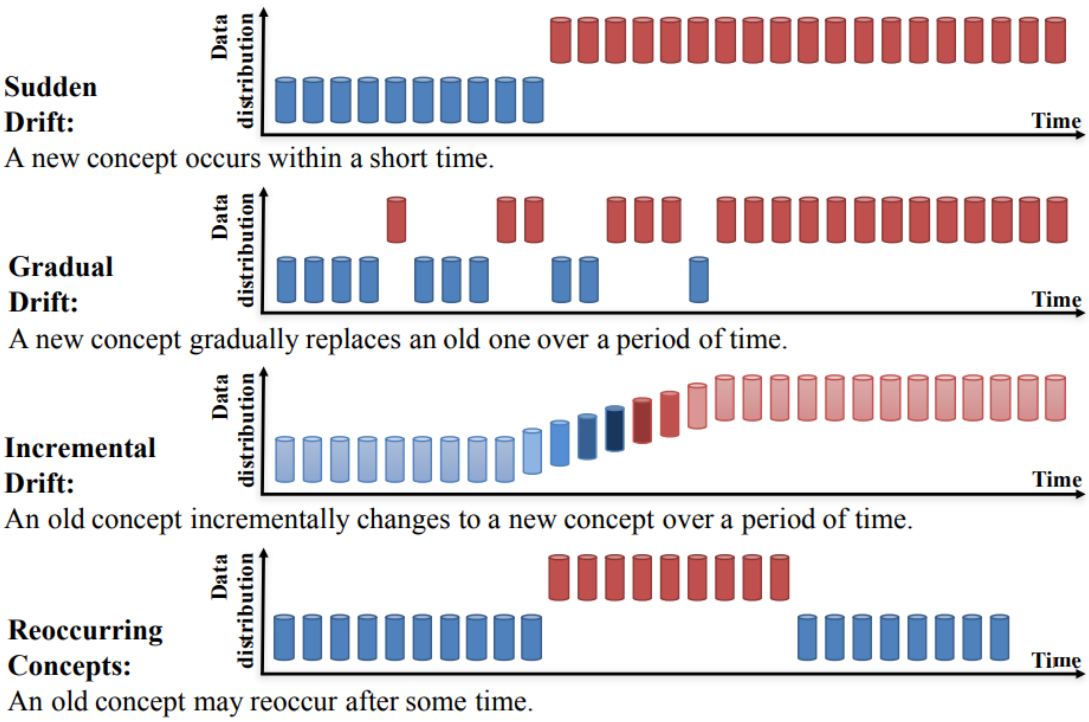


Figure 2.1: *Concept drift types (Žliobaitė, 2010)*

drifts. A detector needs to see a number of instances that represent a new concept before it can detect

the drift. The period from the time when the actual drift occurred to the time when it was detected is called the detection lag (Ang et al., 2013). The detection lag should be as small as possible because the changes in the class distributions can cause the learned model to drop in predictive performance over time (Gama et al., 2004). The sensitivity to this metric is context-dependent. Especially in domains like medical emergency response settings, the detection should happen in real-time. There is no general concept drift detector since different techniques are needed for different kinds of drifts, and each domain requires a different approach (Hu et al., 2020).

Jelsma et al. (2021) created an overview of the currently available concept drift detectors. These drift detection methods can be divided into three categories: (1) error rate-based, (2) data distribution-based, and (3) multiple hypothesis tests based.

Error-based drift detection is done by monitoring fluctuations in error rate over time. The ground truth of the classified instances is needed to achieve this detection. Known methods are Drift Detection Method (DDM) (Gama et al., 2004), Early Drift Detection Method (EDDM) (Baena-García et al., 2006), Hoeffding's inequality-based Drift Detection Method (HDDM) (Frías-Blanco et al., 2015), and Adaptive Windowing (ADWIN) (Bifet and Gavalda, 2007). However, these methods are not successful when labels are costly to get or there is a retrieving delay. Therefore, data distribution-based methods are used in these situations. These methods make it possible to react promptly by monitoring distributional changes in the inputs and outputs of the model. This monitoring is done by checking if the distributions of new data chunks are dissimilar compared to data seen during training. The dissimilarity is quantified using a distance metric and statistically tested for significance. The third method group is using multiple hypothesis tests for drift detection. First, a test statistic and a hypothesis test are applied to detect drift and then these are used to validate the approach. Examples of these are Linear Four Rate and Hierarchical Linear Four Rate (Wang and Abraham, 2015).

The detection approaches can be evaluated if the true concepts are known. This evaluation could be evaluated based on how often they detect drift, how early they detect drift, how they react to it, and how quickly they recover from it. Recent work used a class-based concept drift detection methodology with an unsupervised Silhouette index metric for automatic retraining triggers (Cerquitelli et al., 2019). However, this method lacked a real-world dataset use-case. Baier et al. (2020) researched handling concept drift in Business Process Mining and determined a gap that current research lacks a recommendation on which data should be selected to retrain the ML model. Oliveira et al. (2021) cre-

ated an ensemble adaptive gaussian mixture model to combat concept drift and successfully validated it on synthetic and real-world datasets. Yu et al. (2021) proposed a novel meta-learning framework that learns to classify concept drifts using meta-features, thus it does not rely on humans to judge whether concept drift has occurred. It solves the cold-start problem current detectors have. Jelsma et al. (2021) created a novel detection method and a data generator using data streams with different kinds and different levels of concept drift.

2.5.3 Detection actions

When drift is detected, several different actions are suggested in the literature to combat this drift. These actions are called mitigation mechanisms Gama et al. (2014) researched this and published a survey paper about the adaptations of algorithms on concept drift. These adaptations are dependent on the learning mode of the model. If the learning mode is offline batch learning, then the model can immediately be retrained with data that includes the new concept after detecting drift. However, new labels are needed for this in a classification task. Obtaining these new labels can be costly or too late to be effective if the gathering of the labels is delayed. Another issue is the decision of which instances to select for the new training data. If any of these issues are the case, a more effective approach should be chosen. Another option is switching the learning mode by adapting the model for online learning in which the model will learn continuously. However, online learning comes with its problems. One of the problems is the stability-plasticity dilemma: a trade-off between the ability to preserve previously learned knowledge vs continuous learning. A related phenomenon is catastrophic forgetting, which is the tendency of an ML model to entirely and abruptly forget previously learned information upon learning new information. This phenomenon is not desired in many cases.
Another approach is adapting the model in contrast to only adapting the learning mode. This adaptation can be made by entirely replacing the current model or by making local changes. The challenging task is choosing the right replacement model. Current research defined underspecification as a problem in current ML applications (D'Amour et al., 2020). Underspecification is when there are multiple viable models to choose from with comparable performance. However, some models will be using features that are less prone to drift than other models. A current research gap is that it is challenging to find models or features that are the least drift-prone. A popular example portraying model choice is choosing a weighted ensemble modelling technique. With this, a new model is trained when a new

concept is found. This new model is added to the ensemble. At inference time, the ensemble gives more weight to the latest models compared to the older models. Ensembles can make a difference in combating drift when the drift change type is flexible (Žliobaitė, 2010).

Research need: A current research need is that the focus should switch from researching general drift detection and handling methods to designing more specific, application-oriented approaches. These should address various issues like delayed labelling, label availability, the cost-benefit trade-off of the model update, and other issues peculiar to particular types of applications (Žliobaitė et al., 2016). Another research need in concept drift is about the explainability of drift. The issue with concept drift detection is that current ML models are association mechanisms. It is hard to know when and why concepts change with associations alone because there is no definition of a concept, only covariates that correlate to a concept. More advanced classification techniques are needed to get the best handle on concept drift. At some point, ML systems' output and decision making become too complex for humans to grasp. Detectors based on simple performance measures will not be suitable at that point. Instead of explaining when the change happened, it is more valuable to explain how and why it happened instead (Žliobaitė et al., 2016). Hinder and Hammer (2020) answered this need by creating an efficient algorithm to characterise concept drift based on counterfactual explanations. However, this approach is still limited. A deeper dive into counterfactuals and the causality of drift detection will take place in the Causality chapter.

2.5.4 Influential drift

Influential drift, or prediction drift, is a particular kind of concept drift where the cause of the concept drift is the ML model itself. In contrast to intrinsic drifts, which is caused by the environment inherently changing. Influential drift is a topic in the Influence ML research area, which focuses on the influences ML has. According to the knowledge of the authors, influential drift in ML is a novel topic with scarce scientific literature available. This influence on the environment and the environment influencing the model creates an influential feedback loop. A distinction can be made if the system is self-adapting or not. In a static system, the predictions of the algorithm cause the environment to change. This change will influence the performance of the algorithm. The abstract cause and effect chains of influential drifts are depicted below.

- Static influential drift: Algorithm's predictions → environment change → concept drift → al-

gorithm performance change

An example of a static influential drift can be seen in a spam filter detector. The algorithm is first trained on historical spam messages and these will be detected and not shown to the receiver. The spam creators will change the way they write spam mails to pass through the filter, so new spam concepts emerge. The algorithm will see a decrease in mails that look like historical spam mails, let the new mails through, and thereby showing a high accuracy. However, when the real labels are collected from user feedback about spam mails, the accuracy of the model will drop significantly if it is not retrained.

- Dynamic influential drift: Algorithm's predictions → environment change → concept drift → algorithm performance change → model retrained → environment change → loop

Influential drift can create a feedback loop in a dynamic adaptive algorithm setting. In the spam filter example, the algorithm can be retrained each time new labels are collected and the performance of the algorithm drops to a threshold. By doing this, new spam mails are caught, with a slight retraining delay. Prediction drift only happens when feedback loops are already in place of the model's environment or when the model itself creates and sustains the feedback loop. Because influential drift is responsible for and created by feedback loops, a deeper dive is done into these mechanisms in the following section.

2.6 Feedback loops

A feedback loop occurs when the outputs of one system come back as inputs through a chain of cause and effect that forms a loop. In control systems literature, these are termed closed feedback loops, and in ML literature, this effect is called algorithmic confounding.
The following formal feedback definition is taken from Wager et al. (2013):
Suppose that we have a model that makes predictions $\hat{y}_i^{(t)}$ in time periods t = 1, 2, ... for examples i = 1, ..., n. The goal is is to understand feedback effects between consecutive pairs of predictions $\hat{y}_i^{(t)}$ and $\hat{y}_i^{(t+1)}$. The statistical feedback is defined in terms of counterfactual reasoning: we want to know what would have happened to $\hat{y}_i^{(t+1)}$ had $\hat{y}_i^{(t)}$ been different. To distinguish between counterfactuals, the potential outcomes notation is used. let $\hat{y}_i^{(t+1)}[y_i^{(t)}]$ be the predictions our model would have made at time $t + 1$ if we had published $y_i^{(t)}$ as our time-t prediction. In practice we only get to

observe $\hat{y}_i^{(t+1)}[y_i^{(t)}]$ for a single $y_i^{(t)}$; all other values of $\hat{y}_i^{(t+1)}[y_i^{(t)}]$ are counterfactual. We also consider $\hat{y}_i^{(t+1)}[\oslash]$, the prediction our model would have made at time $t+1$ if the model never made any of its predictions public and so did not have the chance to affect its environment. With this notation, we define feedback as

$$Feedback_i^{(t)} = \hat{y}_i^{(t+1)}[y_i^{(t)}] - \hat{y}_i^{(t+1)}[\oslash]$$

This formula is the difference between the predictions the model made and the predictions it would have made had it not had the chance to affect its environment by broadcasting predictions in the past. Thus, statistical feedback is a difference in potential outcomes. Terms like counterfactuals and potential outcomes used in this definition will be elaborated upon in the Causality chapter.

ML models do not exclusively form feedback loops. For example, in a social influence context, feedback loops already exist. These loops occur because human popularity follows power-law distributions (Mansoury et al., 2020). Research shows that a rating system in the environment can influence the popularity perception of humans. Muchnik et al. (2013) researched how perceiving a rating from a rating system influences humans to rate in the same popularity direction, positive or negative.

While these feedback loops already exist, introducing an ML system in these environments amplifies current feedback loops by increasing the velocity of the feedback process through automation. Therefore, undesired consequences evolve more quickly and increase the amount of impact they can make. Algorithmic bias is termed in literature to describe this feedback loop amplification. Bias amplification in the context of recommender systems can lead to several other problems. These problems are declining the aggregate diversity, shifting the representation of users' taste over time, and also homogenization of the users' experience (Mansoury et al., 2020).

2.7 Feedback loop categorization

Scientific literature about feedback loops in ML settings is scarce. However, an attempt is made to categorize feedback loops from the literature review, including scientific papers, theoretical blogs, conference talks, and articles. A feedback loop can be categorized into several dimensions. First, the dimensions are discussed, which consist of (1) the influence mechanism, (2) the influence direction, (3) the influence consequences, (4) and the influence systems.

1. Influence mechanisms

Often described as positive or negative feedback loops, this terminology can be confusing because of the different usage in different disciplines. For this research in the ML context, it is clearer to describe a positive feedback loop having a self-reinforcing effect and a negative feedback loop having a self-correcting effect. In contrast to looking at if it is a favourable effect (positive) or an unfavourable effect (negative). Favourable looping effects are better described as virtuous cycles and unfavourable looping effects as vicious cycles. A positive feedback loop with a self-reinforcing effect diverges away from the equilibrium. It is inherently a destabilizing mechanism by pushing the environment increasingly further in one direction until it reaches a limit or a self-correcting effect. An abstract mechanism can be described as follows:

$$\text{Increase of X} \rightarrow \text{Increase of Y} \rightarrow \text{Increase of X}$$

A negative feedback loop with a self-correcting effect converges to the equilibrium. It is inherently a stabilizing mechanism by pushing the environment back to a stable state. An abstract mechanism can be described as follows:

$$\text{Increase of X} \rightarrow \text{Increase of Y} \rightarrow \text{Decrease of X}$$

2. Influence direction

An internal way of a feedback loop influence in ML is to influence itself internally through the incoming data in the model. Sculley et al. (2015) state that this internal feedback leads to a form of analysis debt because it is difficult to predict the behaviour of a model before it is released. Their research distinguishes between direct feedback loops and hidden feedback loops. A direct feedback loop causes the selective labelling problem, which is when the model may directly influence its future training data selection, introducing bias (Liu et al., 2020).

An external way of influence is when the feedback loop causes a change in the environment which comes back as data into the algorithm. A direct way of external influence is when the characteristics of a predicted instance are altered because of a prediction. An example of this is predicting a higher interest rate for a credit loan application which is risky, thereby making the applicant riskier. An indirect way is when the prediction of a single instance influences the group where that instance is a part of. This influence can happen through knowledge sharing between instances and their groups (Ang et al., 2013).

3. Influence consequences

Prediction drift and corresponding feedback loops have consequences. These consequences can be wanted or unwanted and malicious or unintended. The two main consequential categories are explained: biased predictions and behaviour change in instances.

Biased predictions:

Biased predictions are created through prophecies of the algorithm. Feedback loops create self-fulfilling prophecies with self-reinforcing effects that have direct influence. A behavioural definition of a self-fulfilling prophecy is a false definition of the situation evoking a new behaviour that makes the originally false conception come true (Merton, 1948). Ferraro et al. (2004) researched how models can become self-fulfilling prophecies where predicting economic growth can cause economic growth by instilling market confidence. In the ML context, this essentially translates to if the model makes a prediction and thereby creates a self-reinforcing feedback loop, then the outcome will be determined primarily by the act of predicting. The predictions of the model push the instances towards its way of predicting.

Self-defeating prophecies are comparable to self-fulfilling prophecies. However, these are created by feedback loops with self-correcting effects. Self-correcting effects are the opposite of self-reinforcing effects. These effects push the instances in the opposite direction of the model's predictions.

Behaviour change:

By introducing an algorithm in the environment, the actors in that environment are likely to change their behaviour if the system has influential power. These behavioural changes are system dependent and will be discussed in the corresponding systems. The main categories are intrinsic changes to the instances and instances trying to game the system.

4. Influence systems:

There are three types of systems that inherently or consequentially create feedback loops. These are recommender systems, classification systems, and monitoring systems.

Recommender systems: Recommender systems can be very influential. The best example of this are content platforms that use recommender systems. The goal of a content recommender system is to recommend content that the user will interact with. Content platforms are virtual places where content that content creators create and content consumers meet each other. These platforms have significant

influential power over what gets shown to consumers since they control the content recommender system. The platform, which is usually a for-profit business, has the goal of profit maximization, usually the time the user spends on the platform. This maximization is done to show the most advertisements in the meantime. Through social influence and network effects, consumers can be influenced to buy or use certain products. This influence becomes problematic when the platform's goals have the effect of being negative for the user. This happens when the recommender systems get over-exploited for KPI maximization since these automate and amplify negative effects. An example of this is maximizing the time the user spends on the platform in such a way that the user spends more time than he wants on it. A behaviour change feedback loop occurs through a self-fulfilling prophecy dictated by the KPI maximization of the institution. The more popular a thing is, the more likely interaction with it will happen, the more popular it becomes. This is called the digital bandwagon effect (Epstein and Robertson, 2015). The content feedback loop looks like:

Increase in popularity $\rightarrow$ increases ranking $\rightarrow$ increases user interaction $\rightarrow$ increase in popularity.

The consequences of this loop are behavioural change of the users and intrinsic change of the content. Homogenization of user preferences is well studied in literature (Chaney et al., 2018). Polarization of opinions happens because nuanced opinions get filtered away since these are not sensational enough to generate popularity (Dandekar et al., 2012). A related effect is the search engine manipulation effect (SEME), where search engine ranking results manipulate user opinions through the trust in the engine (Epstein and Robertson, 2015). Content-wise, the digital bandwagon effect can intrinsically alter the content (Mansoury et al., 2020).

Examples of systems with these loops are the following: Popularity bias in online advertising (Bottou et al., 2013), and social media posts (Muchnik et al., 2013), search engines changing the way news articles are written (The New Economy, 2017), youtube influencing video length (Smith et al., 2018), music recommender systems responsible for the shortening of songs (Bemrose, 2019), and twitter opinion polarization (Conover et al., 2011).

Classification systems: Classification systems which are trained through supervised learning with impactful predictions are prone to creating feedback loops. This is because of increased incentives to get a favourable classification, and thereby a higher cost can be incurred to change features. There are three different scenarios in classification systems that are relevant. This is because, in environments where interactions happen between two actors, both actors can be guided by an ML algorithm.

Each actor has its own goals and effects on the system as a whole. In an environment with multiple algorithms:

- If there is only an algorithm on the classification side, then this is a monitoring scenario where the classification party is trying to catch bad actors in an automated fashion.
- If there is only an algorithm on the acting side, then this is an acting scenario where the acting party is trying to game the classification party in an automated fashion.
- If there is an algorithm on both sides, this is an algorithmic scenario.

Scoring systems: Scoring systems that make important decisions about individuals have the goal to categorize good instances from bad ones (D'Alessandro et al., 2017). This is done under metric optimizations that the organization that owns the algorithm uses. Based on this decision, an allocation feedback loop is created, determining who receives a good treatment and who receives a bad one. The effects of this system are biased predictions through learned historical biases that create self-fulfilling prophecies. This creates incentives for gaming the system through strategic classification.
Examples of these can be found in the following domains: Hiring (Liu et al., 2019), School admissions (Liu et al., 2019), Predictive policing (Ensign et al., 2018), Recidivism (Aneja et al., 2019), Credit-lending (Fuster et al., 2017), Pricing influence (Malik, 2021).
Monitoring systems: In environments where the goal of one party is to catch another party, often with malicious goals, feedback loops occur through learning from automated monitoring systems. The first party is always playing catch up to the second party since the second one always tries to find new methods not to get detected by the first party. These detecting mechanisms on adaptive adversaries create the so-called "cat and mouse game" environments. The goal of monitoring algorithms is to detect specific behaviours. The effects are of two kinds. On the one hand, these systems can alter the behaviour after successful predictions through a self-defeating feedback loop. The behaviours that are monitored will appear less and less. On the other hand, when decisions and actions are made based on an algorithm's predictions that have measured success in the past, it creates a self-fulfilling feedback loop since only successful predictions of the algorithm are acted upon. This is an internal loop compared to sampling bias. The consequences are if this monitoring has potential negative outcomes for the monitored and the monitored has no malicious intent, risk-aversion can be a result. If the monitored party has a malicious intent, then it will try to find new ways of going undetected.

Examples of these can be found in the following domains: Predictive policing (Ensign et al., 2018), Financial statement crafting (Cao et al., 2020), Healthcare predictions influencing outcome (Adam et al., 2020) (Jacobs et al., 2021).

Algorithmic environments: In an environment where multiple algorithmic systems are at play, the interaction between them can create feedback loops. These can be adversarial environments or simply environments where the output of one algorithm influences the input of another. The goal of the adversaries is to make the classifier produce false negatives. The goal of the classifier is to catch the adversaries.

Examples of these can be found in the following domains: Multiple web components (Sculley et al., 2015), Spoofing in algorithmic trading (Goldblum et al., 2020), Malware detection (Galen and Steele, 2020), Competing AI (Ginart et al., 2020), Fraud detection (Dalvi et al., 2004), Financial statements (Cao et al., 2020).

Consequences of classification systems: Influential classification systems cause behaviour change based on the prediction. Regular gaming happens when for example, forecasting a KPI with an algorithm. If the prediction has a positive outcome and behaviour is changed to achieve that possible prediction, then the effect is a self-fulfilling prophecy. This can happen because the prediction increased confidence in achieving the outcome. If the prediction has a negative outcome, and the behaviour is changed to avoid that prediction, then the effect is a self-defeating prophecy. This can happen because the prediction increased the motivation to change the outcome.

Strategic gaming is a different kind of gaming where the acting is deliberate. The systems' imposed classification edges can become self-fulfilling goals through instances changing their features. These feature manipulations will create a drift of joint distributions since the model will see more incoming positive classifications after a period of predictions compared to initial training distributions. For example, in credit lending, if a lender decides that a particular person is a good borrower by providing him with the credit because of his features, then this person could go and tell his friends about this. These friends will learn about the needed feature values to get a positive classification, thereby making it more likely that similar people like the first person ask for a credit and get it. This example has a self-fulfilling prophecy effect through an indirect influence process. Nowak et al. (2018) validated a similar effect in the LendingClub dataset.

Adversarial attacks on the classification systems are also a kind of gaming. However, here the intent

is malicious, and the goal is to get a better classification or degrade the system. This is only possible if knowledge is gained from the system's self-adaptation through online or batch learning by using new incoming data as training instances. If the system self-adapts through the learning of incoming instances, a consequence of adversarial attacks can be adversarial drift. Here, the system is degraded by an accuracy decrease through the learning of misclassified instances.

2.7.1 The need to study influential drift

Influential drift and the created feedback loops in ML are valuable to research for the scientific community because of multiple related reasons: hardening, explainability, and fairness.

Hardening: Current ML systems use a given training dataset to learn a model for that dataset. However, by incorporating learning from change into the learning process, ML systems can become more intelligent and more likely to achieve accurate automated decision making (Chen and Liu, 2018). Lifelong ML system can sequentially learn many tasks from one or more domains in its lifetime and require the ability to retain knowledge, adapt to changes, and transfer knowledge when learning a new task (Madrid et al., 2019). This learning can only happen when the system is robust. For this robustness, gracious handling of change is required, which is the area of hardening ML. A part of the solution to this is accounting for influential drift since the cause of non-stationarity can be the algorithm itself. To account for this drift, it needs to be detected first and handled after that. If the handling action is retraining the model, the prediction influence will still be present, and this retraining will need to happen periodically. If the model is an online learning algorithm, the environment may be pushed to an undesired state due to the inherent influential drift.

Explainability: Together with this hardening for automated decision making, explainability is needed for future ML systems. When influential systems make decisions, by law, these are required to be scrutinized and explained. ML systems designers need to consider how a system influences its users and accounts for algorithmic confounding. Additionally, platform users and policymakers should consider feedback loop effects as they make individual choices or propose policies to guide or govern the use of these algorithms (Chaney et al., 2018). The EU wants AI certification by testing and auditing AI systems in accordance with AI robustness, accuracy, reproducibility, error dealing during all life cycles, resilience to attacks and manipulation (Commission, 2020). It cannot achieve proper auditing without explainability when feedback loops are unaccounted for in the explanation. It should

be possible to explain that an algorithm is not creating unwanted impact through feedback loops or that the feedback loop is known and adjusted for.

Fairness: Through algorithmic bias, feedback loops are amplified by influential drift. The study of algorithmic bias falls under the umbrella of fair machine learning (Abdollahi and Nasraoui, 2018). Existing scholarship on fairness in automated decision-making criticizes unconstrained machine learning for its potential to harm historically underrepresented or disadvantaged groups in the population (Liu et al., 2019). By these accounts, notions of fairness and justice can only be addressed if this constructivist feedback is taken into account (D'amour et al., 2020). However, diversity is often a counterpart to accuracy in classification tasks (Chaney et al., 2018). This skews the incentives for ML makers to account for and monitor diversity in their systems. Liu et al. (2019) state that fairness is primarily studied in static classifications settings without considering how decisions change the underlying population, thereby implying an influential drift of the algorithms. Their temporal regime modelling of fairness criteria highlights the importance to monitor and evaluate these criteria and thereby the related influential drift over time.

2.7.2 Influential drift detection

Research calls for improved drift detection methods. Comprehensive live monitoring of system behaviour in real-time combined with an automated response is critical for long-term system reliability (Krempl et al., 2014). An ideal drift handling system should quickly adapt to drift, be robust to noise, distinguish it from drift, and recognize and treat significant drift in model performance.

Shanbhag et al. (2021) defined three problems with current drift detection approaches relevant to influential drift detection. First, detecting drift in the distribution of individual features may not be sufficient. For instance, it could be that the predictions may drift despite no drift in any of the individual feature distributions because the joint distributions of the features may drift. Furthermore, drift in individual features may not always lead to drift in predictions. For example, this could happen if the drifting feature is unimportant to the model. Finally, detecting drift in the prediction distributions may not be sufficient either. While the prediction distributions may remain the same, it could still be that the input feature distributions have changed in a meaningful way that affects how the model reasons. A challenge of influential drift is that it is not immediate and not observable during training (Khritankov, 2021). This drift only arises after the model is introduced and running functionally.

Then, the detection of this drift is possible by observing the predictions' effect on the environment or the user behaviour. Sinha et al. (2016) deconvolved feedback loops in recommender systems by validating whether a users rating for an item was intrinsic or influenced by the recommender system. However, their results were based on strong modelling assumptions. So far, there are no evaluated methods available to achieve this in scientific literature.

Influential drift detection has many similar challenges and needs compared to concept drift detection. However, in influential drift detection, the timing of the predictions and the created causes need to be taken into account. In the case of indirect influence feedback, the group changes over time through learning about the predictions. Thus this drift depends on its previous predictions and can only be detected after a prolonged amount of time, in contrast to direct influence feedback, which influences an instance instantly through its classification. A way to monitor this is to know what kind of treatment effect classifying such an instance will have. Wager et al. (2013) state that feedback detection is a causal inference problem. A model suffers from feedback if the predictions it makes today affect the predictions it will make tomorrow. Thus there is a need for discovering a causal relationship between today's and tomorrow's predictions. With knowledge about the feedback mechanisms and the detection approaches, specific actions can be suggested to align the algorithm with the goal that it was made for. An understanding of the data generation process is needed to facilitate the detection of influential drift. For this, either a dataset with real drift is needed, or the data must be synthetically generated by a process.

2.7.3 Causal domain differences in drift detection

An understanding is needed of the data generation process behind the ML classifier's data to hypothesize about the drift of an application. In this setting, the problem that needs to be solved can be of two possible classification domains with different causal mechanisms (Fawcett and Flach, 2005). These causal domains can be found in Figure 2.2. Given that X are the features and Y are the labels, Weimer (2019) gives the following examples:

The first domain contains $X \rightarrow Y$ problems. This is where the labels are causally determined by the covariates. Said differently, the values of observables influence the class value. An example of this is where detecting anomalous accelerometer data in cars is done using an average of the last ten measurements as X. Here, recent behaviour determines the anomaly.

The second domain contains $Y \rightarrow X$ problems. This is where labels causally determine the features. Said differently, the class value influences the probability of observations. An example of this is smart alarms in medicine that have to detect when a patient moves using 2 different meters. The movement of a patient can influence one of the meters, and thereby movement Y is causing meter values X.

A new causal domain can probably be introduced with the results of this research: the causal domain of $X \leftrightarrow Y$ problems. When influential drift can be proven, it can be said that the covariates will influence the target. However, the target will be able to influence future covariates. This domain must include the time dimension.

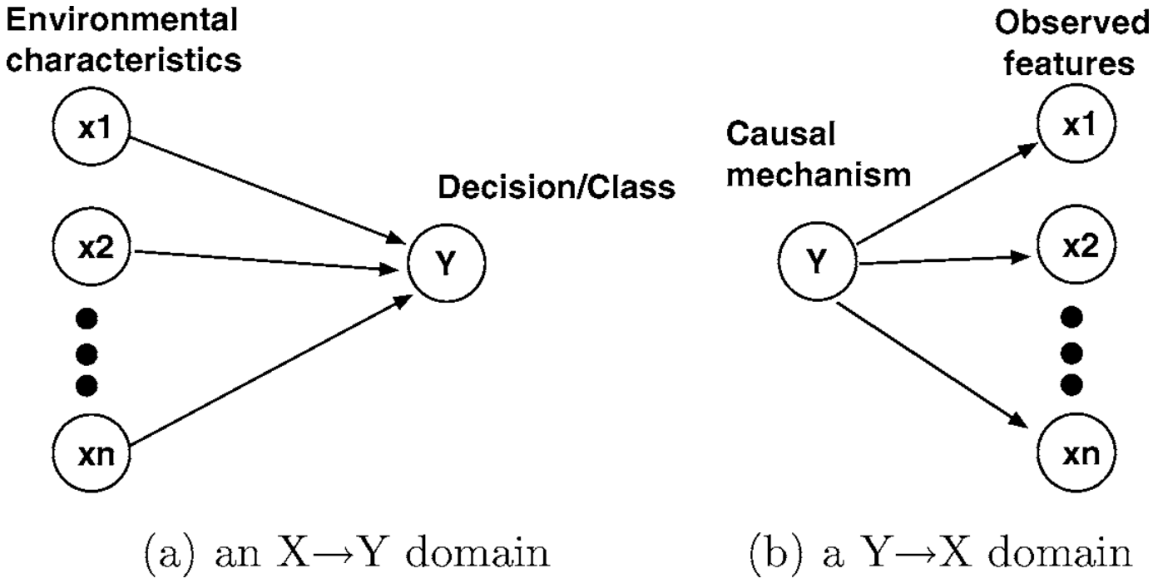


Figure 2.2: *Schematics of two causal domain types (Fawcett and Flach, 2005)*

2.8 Causality

2.8.1 Introduction

Causality is the influence by which a cause contributes to the production of an effect, where the cause is partly responsible for the effect, and the effect is partly dependent on the cause. Causality requires a notion of intervention (Pearl, 2009). Two events often happening together shows a strong correlation between them. However, correlation is not causation. Seeing people with umbrellas outside indicates that it is probably raining (conditional probabilities), however closing umbrellas does not stop the rain (active intervention). This intervention changes the data generation mechanisms and sets a new regime. This regime change challenges the i.i.d. setting, which is assumed in static learning, which is the fundamental problem of concept drift (Scholkopf et al., 2021). Weimer (2019) makes the distinction by calling causality an intrinsic property of the data generation process and correlation a relative property of the data. To establish linear causation, which is the simple direct form of causality like cause X causes Y ($X \rightarrow Y$), three criteria are needed. (1) correspondence, a relationship between the independent and dependent variable, (2) time precedence, a cause must occur before the effect, and (3) non-spuriousness: this relationship must not depend on a third variable which explains both. A statistical dependence can also arise from an unobserved common cause that influences both A and B (confounding) and a common unobserved effect that is conditioned upon in data acquisition (selection bias) (Pearl, 2009).

The best-known method to prove causal statements is doing a randomized controlled trial (RCT). This research method requires a large amount of effort, resources, and robust experiment construction to draw valid conclusions. Experiments need to happen where interventions are done on the data to observe the effect of the intervention to draw causal conclusions. However, often these are not possible due to being too expensive, unethical, or technically impossible to perform (Pearl, 2009). With the absence of a randomization procedure, the relation between cause and effect variables might be confounded. Much effort is put into making causal statements from observational data compared to RCTs.

2.8.2 Need for causality in machine learning

The problems that current ML solutions face were already defined in the influential drift chapter. This section places emphasis on how the causality framework can play a role in tackling these problems.

Hardening of machine learning:

Current ML techniques take in a large amount of data and make predictions based on the correlation or conditional probabilities of features. This approach has shown to be an effective solution to various problems recently. However, there are several issues with these techniques:

Association ML methods stop working in non-stationary environments where the correlations change over time. Old correlations stop being informative for predictions in new regimes, thereby dropping the predictive performance of the trained model. High impact domains like healthcare, law, or financing cannot afford to take risks when correlations are mistaken for causation. There is a need for ML models to be more generalizable, which is the ability to adapt correctly to new and previously unseen data. Causal models are expected to generalize under certain distribution shifts since they explicitly model interventions (Scholkopf et al., 2021).

Robustness should play a role when studying strategic behaviour, adversarial or otherwise. In a credit lending ML context, when one of the model features is the address of the person, a person could decide to move to a better neighbourhood to get a favourable classification. Using a correlation model, this person would increase his chances of getting a favourable classification through conditional probabilities. However, using causal features as input should be more robust to strategic behaviour.

Building further on non-stationarity, these techniques are only able to use the data that was collected. However, the part of the data that could not be collected is missing. Not accounting for this data fails to capture the complete mechanism behind the prediction. For example, in a credit lending context, only outcomes of loans can be collected that have been granted. It is impossible to determine the outcomes of non-granted loans. Some of these non-granted loans might have been successful. An advantage of knowing causal relationships rather than statistical associations is that the former enables the prediction of the effects of actions that perturb the observed system (Mooij et al., 2016).

Explainability of drift:

An entire active research field is currently researching the problem of Explainable AI (XAI) (Barredo Arrieta et al., 2020). In the context of this research, the explainability of drift is the main concern. A

drift detector should alert when a drift occurred and explain why and where it occurred. An answer could be in the form of a list of sensitive features to drift or training data responsible for the drift. This information enables inspection of features where drift manifests itself, enables human understanding of the change, and increases acceptance of life-long learning models. When drift is detected, it is often unclear how to react to such drift (Hinder and Hammer, 2020). This challenge is generally ill-posed and requires expert insight. Thus, an explanation would enable a human to initiate an appropriate reaction. Hinder and Hammer (2020) created an algorithm to characterise concept drift based on counterfactual explanations. However, it is only usable for discrete-time points with well-defined drift. An extension is needed for continuous drift.

A solution for this could lay in transitioning from correlation models to causal models. When modelling the causal structure of the problem, the cause and effect factors affecting a prediction are explicitly revealed. Thereby, through transparency, Causal AI is intrinsically explainable and interpretable. For example, a causal explanation could answer if the drift is influential or if the drift is caused by an adversary trying to attack the model (Sethi and Kantardzic, 2018).

Fairness of ML:

D'Alessandro et al. (2017) show that causality is needed to make current opaque algorithms fair. By law, the court typically requires a causal connection between a decision and the affected person's membership in a protected class to prove discrimination. D'Alessandro et al. (2017) call for causal inference to be used as auditing tools for ML models to access and monitor for fair treatment by measuring metrics like disparate treatment and disparate impact.

A need for causal fairness can be found in the research of Liu et al. (2019). They researched fairness in automated decision-making systems and described the different regimes in which one fairness criterion is preferable over another. To achieve this, they call for an understanding of the two-variable causal mechanism that translates decisions to outcomes.

Quantification of algorithmic effects:

Combining AI explainability and counterfactuals provides the means to not only explain the prediction but, in addition to that, also explain what would have happened if the instance had different covariates. First, this counterfactual information can quantify the delta change needed for a different classification. Consequently, after the different classification is received, the effect this has on the instance should also be quantified to get a more extensive view of the algorithmic influence. This

effect can be measured by calculating the treatment effect of the algorithms' predictions.
Chaney et al. (2018) studied algorithmic confounding in recommender systems and found that the created feedback loop causes homogenization of user behaviour. This effect is a prediction treatment since the recommendations of the system change covariates of the instance, thereby creating a feedback loop. They propose to apply causal reasoning techniques to counter the effects of algorithmic confounding.

2.8.3 Causal ML trend

By realising the severe theoretical limits on the power and performance of current statistical machine learning systems, a need for different tools arose. Judea Pearl, the creator of Bayesian Networks and the renowned frontrunner of the Causal AI field, advocates for causal models in ML. For the next level of improvement of ML algorithms, these algorithms need to have a model of reality similar to the ones used in causal inference (Pearl, 2018). Learning a causal model from the data in the field of artificial intelligence was already a notable topic 30 years ago (Pearl and Verma, 1995). The language of structural causal models is a formal framework for causality (Pearl, 2009). This language consists of structural equation models (SEMs), which are used to learn the causal structure of the data. A graphical notation in the form of a directed acyclic graph (DAG) is added to create a causal graph with these models. The nodes of these graphs denote a variable, and edges denote a causal relationship between two nodes. The skeleton of the causal graph is the graph without directed edges. Causal graphs are used to describe the process of data generation and are usually modelled to provide transparency about the assumptions made for that process.
An intervention is when a causal structural equation is modified by a new structural equation. By doing this in a causal structure, other variables can also change. This doing of an intervention on data has received a new operator, do(·) (Pearl, 2009). Instead of talking about conditional probabilities like P(X=x|Y=y) where the data is only observed, do calculus allows talking about P(X=x|do(Y=y)), where Y is set to y by the researcher. The tools offered by this new calculus can be used to calculate interventional effects though deconfounding of unobserved variables. Examples of causal relationship graphs can be seen in Figure 2.3 taken from Mooij et al. (2016).

While current ML models are of causality level 1, and interventions are of causality level 2, counterfactuals are on level 3 of causality. These can give answers to hypothetical model scenarios. For

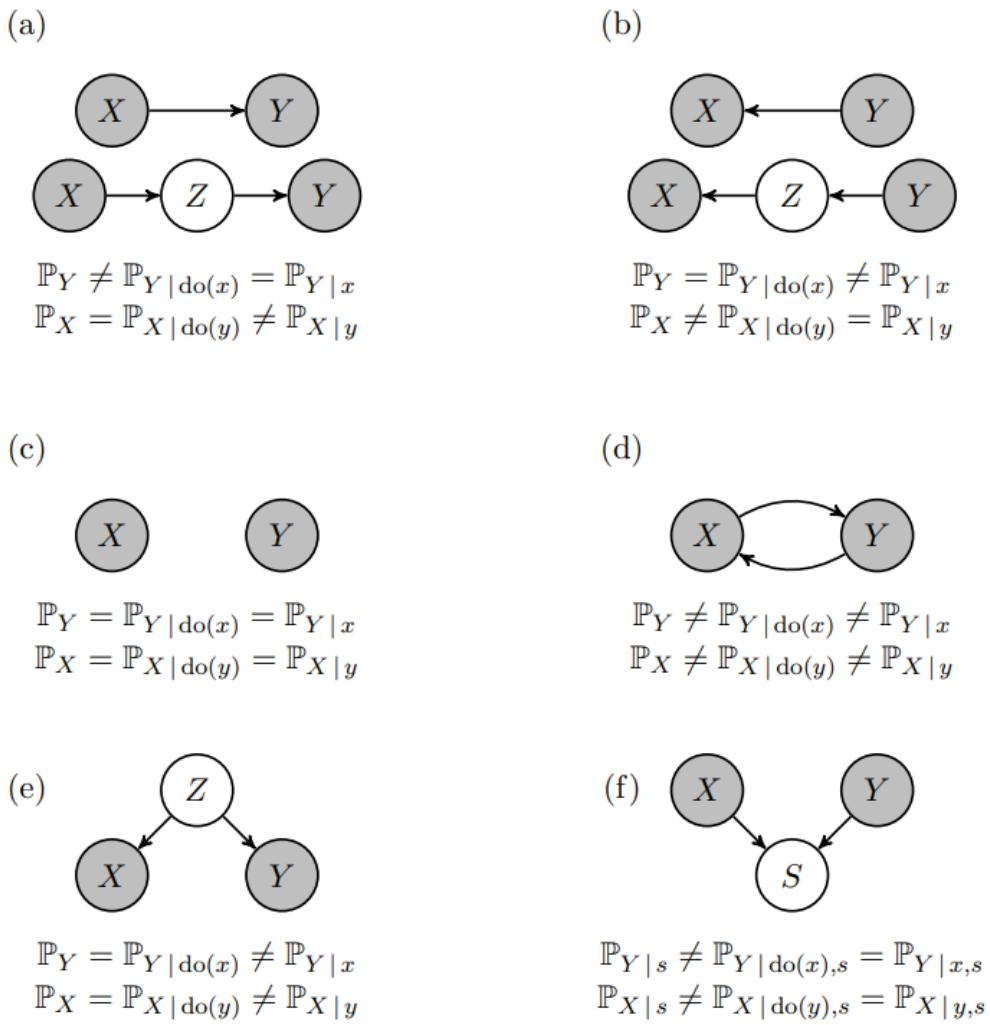


Figure 2.3: *Several possible causal relationships between two observed variables X, Y and a single latent variable: (a) X causes Y ; (b) Y causes X; (c) X, Y are not causally related; (d) feedback relationship, i.e., X causes Y and Y causes X; (e) a hidden confounder Z explains the observed dependence; (f) conditioning on a hidden selection variable S explains the observed dependence (Mooij et al., 2016)*

now, this is out of the scope of this research. The causal hierarchy in Figure 2.4 shows what type of questions can be answered using the causality.

Applying causality to ML is done by changing the learning paradigm of algorithms. Instead of resting on the traditional i.i.d. assumption, a weaker assumption is made that the data on which the model will be applied comes from a possibly different distribution but involving (mostly) the same causal mechanisms. Recently researchers started to apply this change in their research. Examples of this are seen in strategic classification by Miller et al. (2020) who used a causal model together with an agent model to model incentives in an ML context or in recommender systems where Zhang et al. (2021) applied causal interventions to learn about popularity bias effects.

Level (Symbol)	Typical Activity	Typical Questions	Examples
1. Association $P(y\|x)$	Seeing	What is? How would seeing X change my belief in$Y$?	What does a symptom tell me about a disease? What does a survey tell us about the election results?
2. Intervention $P(y\|do(x), z)$	Doing Intervening	What if? What if I do X?	What if I take aspirin, will my headache be cured? What if we ban cigarettes?
3. Counterfactuals $P(y_x\|x', y')$	Imagining, Retrospection	Why? Was it X that caused Y? What if I had acted differently?	Was it the aspirin that stopped my headache? Would Kennedy be alive had Oswald not shot him? What if I had not been smoking the past 2 years?

Figure 2.4: *The Causal Hierarchy. Questions at level i can only be answered if information from level i or higher is available (Pearl, 2018)*

Current research in causal AI is focused on the following two paths: The first path is learning the causal structure of the problem. The second path is predicting from observational data.

1. Causal discovery: Most state-of-the-art causal discovery algorithms that attempt to distinguish these cases based on observational data require that X and Y are part of a larger set of observed random variables influencing each other (Mooij et al., 2016). To aid with causal discovery, observing a system in different environments or contexts can significantly help identify the causal structure (Scholkopf et al., 2021). These contexts can come from interventions, nonstationary time series, multiple views, or interpreted as different tasks like meta-learning. Several algorithms are available for the causal discovery task, classified into two families of methods. The first family is based on Additive Noise Models, and the second family is based on Information Geometric Causal Inference (Mooij et al., 2016). However, these methods are computationally expensive and have the problem that different causal models can generate the same statistical pattern recognition model (Scholkopf et al., 2021). The performance of these methods has to be improved further in order for them to be practical.

Assumptions: To reason about the causal structure, several assumptions need to be taken into account:

Faithfulness: One can sometimes recover aspects of the underlying causal graph from subsets of observed variables in observational data by performing conditional independence tests (Pearl, 2009).

Another set of assumptions are those made on SEMs. While these can be estimated, explicit assumptions about their structure must be made for valid causal discovery.

Possible simplicity assumptions need to be made to make the problem tractable. These can include

no confounding, no feedback loops, no selection bias (Mooij et al., 2016). However, for influential drift detection, these feedback loops are needed to be present.

2. Causal inference: Causal inference is a subfield of statistics that deals with cause-effect relationships. It attempts to identify the causes of a phenomenon by establishing covariation of cause and effect, a time-order relationship with the cause preceding the effect, and the elimination of plausible alternative causes (Shaughnessy and Zechmeister, 2015). The causal inference framework can be used to describe causal effects in non-randomised and observational data. However, the methods used in this subfield rely on heavy assumptions and on the knowledge of the data generating process behind the data. The methods will be discussed in-depth in a later section.

A challenge with causal inference are variables that affect both the exposure of items and the outcomes. These variables are called confounders. Validity problems can occur when observational data is modelled without taking confounders into account.

Challenges with Causal ML:

Scholkopf et al. (2021) mentioned the following serious challenges with causality in ML: (1) inferring abstract causal variables from the available low-level input features, (2) no consensus on which aspects of the data reveal causal relations, (3) train-test protocol may not be sufficient for causal relations and new benchmarks need to be created, (4) even in the limited cases that we understand causality in ML, we often lack scalable and numerically sound algorithms.

2.8.4 Causality & feedback loops

So far, the causal topics discussed were mainly about linear causality. However, in this research focussing on feedback loops, we are not merely interested in linear relationships. When considering feedback loops, non-linear causality should be taken into account. Non-linear causality is when the effect can also cause the cause through feedback, thus $A \leftrightarrow B$. This non-linearity can lead to self-reinforcing processes or divergent outcomes like the butterfly effect. This non-linear causality can happen between levels in a system, for example, a bidirectional flow between micro and macro scales, and between time where bidirectional flows happen between past and future. An example of time can be that a set future goal can feedback on current events.

When using causal discovery to discover SCMs in a feedback loop context instead of using DAGs, which are acyclic, other graphs need to be used that allow for cyclic edges. An example of this was

given by (Mooij et al., 2016) in Figure X. The causal link (d) is described as a feedback relationship. Here, The cyclic relationship is one where X causes Y and Y also causes X. An example of such a relationship can be found in climate change. Rising global temperatures cause the sea ice to melt, which in turn causes the temperature to rise because the ice reflects more sunlight.

Related research: Most research applies assumptions that assume no feedback in the causal diagram of the problem. By removing this assumption, the problem becomes much more complex. The researchers that do loosen this assumption mostly focus on the theoretical aspects of causality in a cyclic setting. Voortman et al. (2010) take the temporal nature of dynamic systems into account where feedback loops occur. They created a causal discovery algorithm for time-series data and proved that it could identify instantaneous feedback loops under common assumptions. This identification makes the model more useful for predicting the effects of manipulating variables when the system is in equilibrium. Other econometric methods like Granger Causality, Vector Autoregression, and Dynamic Bayesian Networks do not have this feat.

Recently the cyclic causality literature became a more active area. A more recent approach showed that the FCI algorithm could be applied in a cyclic setting under the assumptions of Markov property, Faithfulness, and absence of selection bias (Mooij and Claassen, 2020).

Another related paper that uses causality in a feedback loop setting comes from Liu et al. (2020). They applied interventions on the causal model of strategic gaming in an ML setting to measure the impact on fairness of several groups. They showed that common properties of real data lead to undesirable outcomes.

Causal AI in this research: Since a prediction of the algorithm can be modelled as a treatment, a treatment effect can be calculated using causal do(·) calculus. By calculating this treatment effect using causal inference, it is possible to quantify the direct influence the algorithm's prediction will have on the individual instance or the indirect influence on a particular group. The methods and tools available for causal inference on observational data are discussed in the next chapter

2.8.5 Causal methods for observational data

Since RCTs are not always possible, multiple methods are used in practice to mimic RCTs. The main goal of these methods is to estimate the Average Treatment Effect (ATE). This ATE is the difference in the treatment group's effect compared to a control group. These can be used to answer

counterfactual questions for an individual like “What would have happened if I took the treatment?” vs “what would have happened if I did not”.

A/B testing

Called controlled experiments by literature, these tests provide a methodology to reliably evaluate ideas to test for causal relationships by neutralizing confounding variables through randomization assignment (Kohavi et al., 2009). In these tests, users are randomly exposed to one of the two variants: Control A or Treatment B. Each test group gets a different view of the webpage, and metrics are collected for their usage in real-time. This testing is useful to solve the missing data problem to collect data in a new environment. A/B testing is useful when labels are instantly gathered, data is plentiful, and costs to randomize are low. This methodology is the web variant of RCTs. However, RCTs are mostly done on a small scale. Participants know they participate in a study, in contrast to A/B tests done on a large scale. Here, the users unknowingly participate by getting a random version of the web page and getting monitored. A/B tests are a common method used by most major IT firms to improve the pages on the web and algorithm results.

Assumptions and limitations: These tests need to be crafted beforehand with a great amount of care. A/B testing might not be viable if offering a different version is costly. These tests usually run for a predefined period or until enough data is gathered for statistical significance. Thus, the feedback loop can be long and costly. The validity is challenged if anything other than the variable you are testing for is different in the two experiments. In addition to that, certain biases can steer the results. The newness effect can make the change seem significant, even though the user acts differently just because something is new instead of better. Vice versa for older users who dislike the change and become less efficient with the webpage (Kohavi et al., 2009).

Examples: Netflix uses A/B testing before rolling out a new version of their recommendations algorithm (Gomez-Uribe and Hunt, 2015). The main purpose is to compare user engagement with Netflix and member cancellation rates across variants of the algorithm. They call the interpretation of A/B tests an art. Sometimes significant global results are paired with no local changes and vice versa. These tests are then repeated to check the validity.

Slight randomization

Another possible way to measure if there is feedback or impact from a decision is using slight randomization. This can be done by inserting an artificially small amount of noise into the predictions

made by the model. After that, the effect of this noise in future predictions is measured. If those predictions change when noise is added, then the system has feedback. This method is not common and mainly used in research. Proper knowledge of analysis methods is needed to use this method right.

Assumptions and limitations: The central assumption for this test is that the slight randomization can be detected in the later predictions. Problems arise when the costs to add the noise are high, or validity issues arise when the consequence of the noise addition is significant. **Examples**: Wager et al. (2013) tried detecting feedback loop drift to get causal relationships between previous and later predictions. They did this by introducing a local randomization scheme by adding noise to the predictions. Yu et al. (2021) achieved above state-of-the-art results using an extension of the Influence Function to measure the effect on an estimator when adding a small perturbation on one training sample. Their result could reveal the importance of one training sample on the performance of a recommender system.

Rubin's potential outcomes framework

This framework is quasi-experimental since its methods do not make interventions on the data like the other methods. Instead, it leverages the information of large observational data together with strong assumptions on the data generating process to draw estimations on conclusions. This practice is called observational causal inference (Rubin, 1973).

Method: Inverse Propensity Score (IPS) matching.

This method views causal inference as a missing data problem and assumes that the subject characteristics influence the treatment selection in observational studies (Rosenbaum and Rubin, 1983). IPS is a state-of-the-art technique based on machine learning which tries to mimic an RCT by reducing the bias in treatment effect estimates. The main challenge is that there is no data for a non-treatment outcome for a treated person. A classifier is trained to find a likewise individual who was not treated, thereby creating an artificial control group to tackle that challenge.

Propensity score: The core principle of IPS is to estimate the propensity score. The propensity score can be interpreted as the probability of treatment assignment conditional on baseline characteristics. This score is used to reweight the training samples such that the mean of the training distribution after reweighted is equal to target unbiased distribution Yu et al. (2021). By doing this, the characteristics between the treated and the non-treated should be balanced and adjusted for bias selection. Research

suggests that Logistic regression or Neural networks can be used to estimate the propensity score (Hassanpour and Greiner, 2019).

Assumptions and limitations: An accurate IPS estimator should have strong ignorability, which includes two assumptions (Rosenbaum and Rubin, 1983). The first one is unconfoundedness, which assumes there are no unobserved confounders in the data. The second assumption is overlap, where each instance has a non-zero chance of being assigned to any treatment (Hassanpour and Greiner, 2019). If the data has no overlap, the propensity score is non-informative. Other issues are that the data needs to be large enough to increase the possibility of finding an artificial control group. Selection bias in observational datasets implies having fewer instances within each treatment arm at specific regions of the domain (Hassanpour and Greiner, 2019). This bias results in a decrease in the accuracy and confidence of the estimations.

Examples: Researchers of Uber proposed an adversarial validation approach to tackle concept drift using inverse propensity score matching to check whether new data belongs to the test dataset (Pan et al., 2020). If concept drift is detected using this method, the model is retrained before making an inference. Matching on propensity score is more efficient than on the joint distribution of all confounding variables.

2.9 Tools and frameworks

Several programming tools, packages, and frameworks were collected from the literature review. These tools are compatible with Python and can be used for the implementation of the detection approach of this research.

Drift detection: River was mentioned in the introduction section as a package for online machine learning with streaming data that makes use of datastreams. Several drift detectors are available in this package (Montiel et al., 2021). D'amour et al. (2020) advocate for using simulation as a tool for assessing the long term dynamics of ML decision systems. They propose ML-Fairness gym, a new simulation method that allows for explorations of counterfactuals, system-level dynamics, feedback loops, and other long term effects (D'amour et al., 2020). Alibi-detect is yet another open-source library with drift detectors for TensorFlow and PyTorch (https://github.com/SeldonIO/alibi-detect).

Data generation functions: These generators are mostly available in the River package and depend on what data is needed. SEA generator, Rotating or Moving Hyperplane generator, AGRAWAL

generator, RTG and RBF generator

Causality: Microsoft's DoWhy package (https://microsoft.github.io/dowhy/), Causalinference (https://pypi.org/project/CausalInference/), Pymatch (https://github.com/benmiroglio/pymatch)

Chapter 3

Methodology

3.1 Theoretical method background

The goal of the method:

To create a detection approach that can detect prediction influence drift given a dataset that has instances, their corresponding features and feature values, and the classifications that a classifier made. After analyzing this data, the detection approach should provide insight into the influential power of the algorithm by answering the following questions:

- Detection: Is there influence happening?
- Classification: What kind of influence is happening?
- Quantification: How strong is the influence?
- Timing: How long does it take for the influence to manifest itself?

Detection environment differences:

Stationary environment: In a stationary context, the environment does not change. Through time, the same corresponding feature values of instances are causing the same outcomes of those instances. The high-level causal model is shown in Figure 3.1.

$$P_{t=0}(Y \mid X) = P_{t=1}(Y \mid X)$$

Figure 3.1: High level structural causal model of a stationary context

Feedback loop environment: In a feedback loop context, instance outcomes influence instance features over time. These cause the instance features to change, which in term causes the instance outcomes to change. The high-level causal model can be seen in Figure 3.2

$$P_{t=0}(Y \mid X) \neq P_{t=1}(Y \mid X) \; and \; P(Y) \neq P(Y \mid do(X)) \neq P(Y \mid X)$$

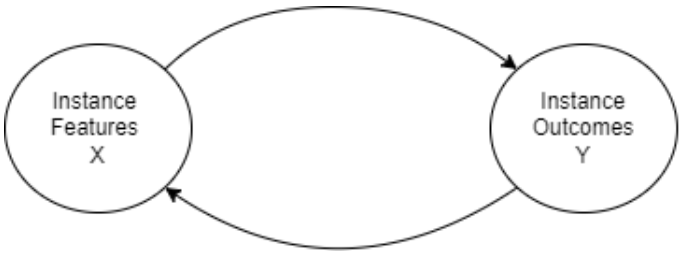


Figure 3.2: High level structural causal model of a feedback loop context

Prediction influence environment:

The shown feedback loop context in Figure 3.2 can be caused by the prediction of a machine learning model. When this machine learning classification is used as a confounder of the feedback loop, the loop disappears. Thus, the model's predictions need to be related to the instance features and outcomes over time to spot and quantify such feedback loops. An interventional study needs to be done on the data to achieve this. For the perfect detection method, a significant portion of the data must be assigned to a treatment group (predicted instances) and the other to a control group (random instances). All the confounders of the treatment should be accounted for. By measuring the change of treated vs control, the average treatment effect (ATE) would quantify the indirect influential drift, and the individual treatment effect (ITE) would quantify the direct influential drift.

However, it is not realistic to do an interventional experiment on all the instances that the model classifies. Additionally, applying such a method on observational data does not work since there is no

way to intervene on W in history.

$$P_{t=0}(Y, X \mid do(W)) = P_{t=1}(Y \mid X) \; and \; P(Y) = P(Y \mid do(X)) \neq P(Y \mid X)$$

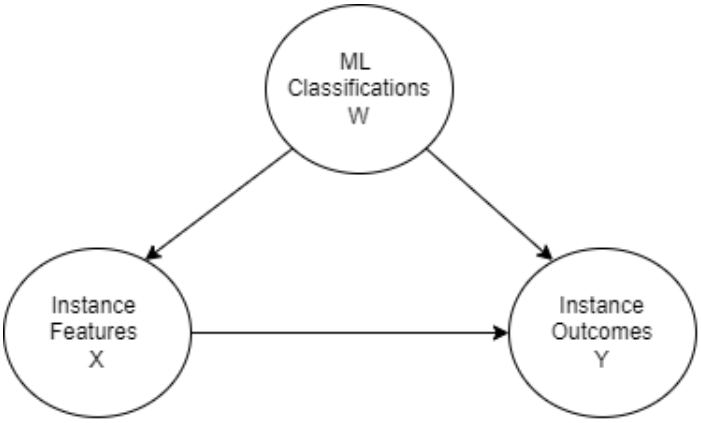


Figure 3.3: High level structural causal model of a prediction influence context

Feedback loop details

Indirect self-fulfilling feedback loop dissection: This research will focus on detecting influential prediction drift that manifests from an indirect feedback loop. An example of such a loop is given in Figure 3.4. The counterpart to this diagram is the self-defeating feedback loop, where the only difference is in the looping from metric to environment.

The explanation of the left diagram: The left side of the graph shows a standard ML prediction flow. The environment is the context in which the machine learning algorithm is predicting instances. In an online-learning feedback loop setting, new instances arrive into the ML algorithm that need a classification. These instances are produced by the environment. When these instances arrive in the ML model, they get a classification. In a binary setting, this is a positive or a negative classification. However, the model does not yet know the true class of the label. This label arrives at a later time point. The lag is the time between the prediction and the arrival of the ground truth. After this ground truth is collected, the prediction is compared to the true label. The model and the performance metrics are then updated with the new sample.

Self-fulfilling loop: When a self-fulfilling feedback loop is present in this scenario, the environment is notified when a model prediction on an instance occurs. The effect of this is that the envi-

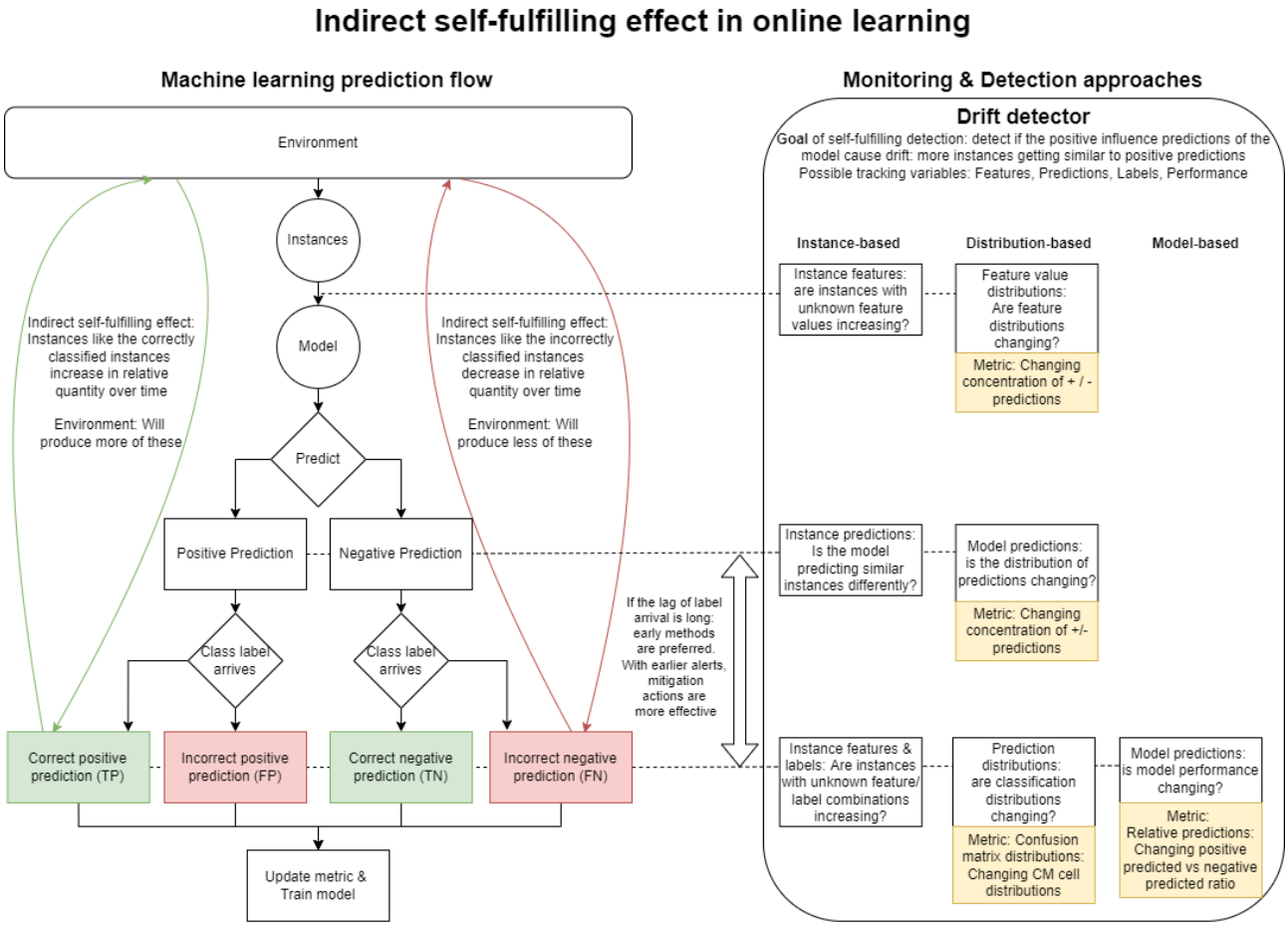


Figure 3.4: Detailed overview of self-fulfilling feedback loop and the corresponding monitoring and detection methods

ronment starts to change. If the correction was positive, more instances like the correctly classified instance will use the model—Vice versa for the negatively classified instances. Less of the negatively classified instances will use the model. In contrast, an indirect self-defeating feedback loop would cause the relative decrease in quantities through a correct classification and vice versa, the relative increase in the quantities of incorrectly classified instances. The environment will produce less of the correct classified ones and more of the incorrectly classified ones.

Loop causes: Two possible causes are found in literature which can be responsible for creating loops in an environment. In a direct individual feedback loop, instances can apply strategic gaming to get a positive classification from the model Liu et al. (2020). This structural causal model is visualized in Figure 3.5. This influence effect assumes that instances can be classified by the algorithm multiple times and that the reward of being positively classified outweighs the cost of changing the feature values.

Another cause of the feedback loop can be through an indirect influence. This can happen through the network sharing effects. This causal model is visualized in Figure 3.6

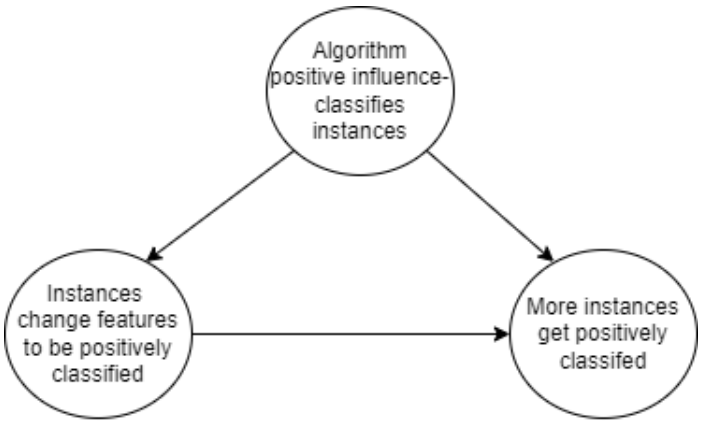


Figure 3.5: Strategic gaming: direct feedback loop through algorithm classifications

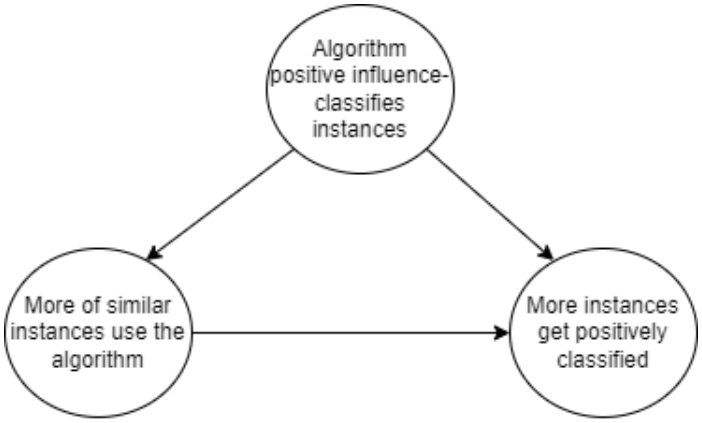


Figure 3.6: Network effects: indirect feedback loop through algorithm classifications

Model evaluation: This feedback loop will be noticeable in the performance metric of the model. More instances like the correctly classified one will arrive. Therefore, the model will better know how to classify these based on its historical performance, thereby reinforcing the model. The number of positive classified instances will increase relative to the decreasing number of negative instances. Because of this effect, the performance metric will increase until a maximum when only positive instances arrive

Monitoring and detection: On the right side of Figure 3.4, the different monitoring and detection approaches are mapped onto the ML prediction flow and classified into three different categories: instance-based, distribution-based, and model-based. Each approach has its pro's and cons in a drift detection pipeline. However, only one of them is chosen for the implementation of the prediction influence detector. They also differ in timing when the metric changes based on the ML prediction timeline. The detailed explanations of the other methods will not be given since these are not relevant for this research.

Chosen detection method: The detection approach that is implemented is a distribution-based

approach that uses the predictions of the model. Even though an early detection approach would result in earlier detections, it is chosen to use the true class labels of the instances. This is chosen because timing is not considered in the experiment that this approach is based on. In addition, a detection approach that only uses the predicted class labels will result in less robust drift conclusions than an early detection approach. This research is a follow-up of the research of Jelsma et al. (2021). They used the true class labels of the instances to construct a confusion matrix. Each cell of this confusion matrix was tracked for changes over time, and a linear regression model was fit to predict the future changes. However, this did not result in satisfactory detection results. Both approaches' differences are shown in Table 3.3. The detection approach of this thesis will differ from this method by using the structural causal model hypothesis shown in Figure 3.6. The correct vs incorrect classifications, which are the W according to the figure, will be related to the distributional changes of the data. Randomization interventions are made on a small percentage of the data such that changes in the outcomes can be detected and quantified. This will then be related to the feature values through trials over time. This method is in accordance with the slight randomization section in Section 2.8.5 and is comparable to the study of Wager et al. (2013).

Conceptual method:

Experiment planning of quantitative experiment on a simulated systematic classification environment with indirect prediction drift detection through A/B testing randomization:

Research type: applied quantitative synthetic causal research

Goal of approach: The goal of the created detection approach is to detect, classify, quantify, and localize prediction influence drift through the use of randomization.

A side goal of the method: Create a checkerboard prediction influence dataset.

The randomization of the approach should account for the use of such methods in real-world cases. It can be a costly business endeavour to classify instances differently than the classifier is trained for. Thus, the aim is to use as little randomization as possible to validate the drift detector.

Experimental context selection: The experiment will be an online general synthetic data problem with a between-subject experimental design. If the prediction influence effect is proven in this setting, then a generalization should be made to real data with comparable characteristics.

Experimental hypothesis:

H0: No real underlying pattern between classifications and instance feature distributions over time.

H1: Classifications cause a pattern in instance feature distributions over time.

Experimental Variables:

Independent variable: Systematic model classifications of instances based on their feature values.

Dependent variable: Relative kernel density estimation changes percentages of the instance's feature value compared to the future feature space KDE.

However, this dependent variable can also be changed through third factors. It is important to prove that the density change of the instances is not caused by randomness but by the classification of the algorithm. Therefore, random regime shifts are applied to make this distinction more clear.

Sampling methods: The treatment group is selected based on a checkerboard classification pattern with regime shifts. The control group is selected within this regime based on a pseudorandom flip percentage. The samples are analyzed per true instance classification. This is done to make a drift conclusion per true class since each class can react differently to a classification because of the inherent nature of the classification, which can have a positive outcome for one class, and a negative outcome for the other class. A sample needs to be selected for the before and after KDE calculation for the drift quantification. Per true class, based on the feature spacetime trial of the instances, samples are chosen in the first period of the trial. These samples are then compared to the end period of the same trial.

Formalization of experiment: The distributional based prediction influence detector should calculate the distribution change of the joint probability of features and predictions over time:

$$P_{t=0}(X, X) \neq P_{t=1}(X, Y)$$

A drift type classification can only be determined when the classification outcome is considered in the change calculation. This outcome can be correct or incorrect. For a positive class, this is a positive prediction. For the negative class, this is a negative prediction.

$$Correct\ outcome: \ P(X, Y = + \mid Y_{pred} = +) \ \& \ P(X, Y = - \mid Y_{pred} = -)$$

$$Incorrect\ outcome: \ P(X, Y = + \mid Y_{pred} = -) \ \& \ P(X, Y = - \mid Y_{pred} = +)$$

Because this calculation happens per true class of the instances, only one joint probability of the correct outcome and one joint probability of the incorrect outcome is used for the calculation.

$$Correct\ outcome\ class +:\ P(X, Y = + \mid Y_{pred} = +)$$

$$Incorrect\ outcome\ class -:\ P(X, Y = - \mid Y_{pred} = -)$$

In a self-fulfilling prophecy (SF), the correct joint probability of t+1 is bigger than the correct joint probability of t=0. Vice versa for the incorrect joint probability. This should be smaller. For class +, these are:

$$Correct\ SF\ prophecy\ class +:\ P_{t=0}(X, Y = + \mid Y_{pred} = +) <\ P_{t=1}(X, Y = + \mid Y_{pred} = +)$$

$$Incorrect\ SF\ prophecy\ class +:\ P_{t=0}(X, Y = + \mid Y_{pred} = -) > P_{t=1}(X, Y = + \mid Y_{pred} = -)$$

In a self-defeating (SD) prophecy, the correct joint probability of t+1 is smaller than the correct joint probability of t=0. Vice versa for the incorrect joint probability. This should be bigger. For class +, these are:

$$Correct\ SD\ prophecy\ class +:\ P_{t=0}(X, Y = + \mid Y_{pred} = +) >\ P_{t=1}(X, Y = + \mid Y_{pred} = +)$$

$$Incorrect\ SD\ prophecy\ class +:\ P_{t=0}(X, Y = + \mid Y_{pred} = -) < P_{t=1}(X, Y = + \mid Y_{pred} = -)$$

This change calculation will be done per trial. t=0 is the start of trial T, and t=1 is the end of trial T.

$$Trial\ joint\ probability\ class +:\ P(X, T, Y = +)$$

The calculation will be done using relative change. The relative change from in trial T is:

$$Relative\ change\ T\ class +:\ P(X, T, Y = +) = (P_{t=1}(X \mid Y = +) - P_{t=0}(X \mid Y = +))/P_{t=0}(X|Y = +)$$

Table 3.1: Fixed parameters of detection approach

Component	Parameter	Value	Explanation
Experiment	Classification method	feature_grid	This is the checkerboard (grid) pattern that gets created onto the feature space.
Streams	Feature_slice_length	1	The length of the feature space
Streams	Min_feature_start	0	The minimum value of the feature to be analyzed.
Streams	Max_feature_end	2	The maximum value of the feature to be analyzed.
Streams	N_classes	2	Number of classes the stream produces.
Streams	N_features	1	Number of features each instance of the stream has.
Streams	N_centroids	1	Number of centroids the stream has.
Streams	Std_dev	0.1	The standard deviation of the gaussian from where feature values get generated from the centroid.
Dataset	Instances_per_day	10	The number of generated instances per day.
Detection	Statistical_test	mannwhitneyu	The statistical test used for influence drift detection.
Detection	Alpha	0.01	The alpha value used for influence drift detection.

Table 3.2: Variable parameters of detection approach

Component	Parameter	Range	Explanation
Experiment	Max_samples	100-100000	For how many instances the experiment will last.
Experiment	N_trials	0-10	By how many flip trials the max_samples should be split.
Experiment	Flip_percentage	0.0-1.0	The percentage amount of instances predictions to flip
Dataset	Prediction_influence_drift	Self-fulfilling, self-defeating, and no-influence	What kind of influence classifications create on the environment.
Dataset	Influence_multiplication	0.001-1	The strength of the influence drift.
Detection	Change_calculation_method	Single_instance_density_ partial_feature_space, single_instance_density _full_feature_space	Different method options for the within trial change calculations.
Detection	Detection_method	Classification, absolute_majority	The method for post-processing a significant statistical test.

3.1.1 Method implementation

The above method is implemented through the use of the River python package Montiel et al. (2021). Each component used in the experimental setup is elaborated upon, and the inner workings are explained. Because of the modular setup of this experiment, each component can relatively easily be switched out for another version of the same component.

Environment setup

Since getting real-world data with a validated feedback loop is hard, it is chosen in this research to create a synthetic dataset. A dataset is simulated, which contains a manufactured feedback loop to validate a detection approach on. Combining this dataset with a classifier in a question and answer session created the simulated environment where the detection approach should detect prediction influence drift. This all is done through the following components:

Data stream: Random Feature Radial Basis Function (RFRBF) generator

The RFBF generator is a custom Random RBF (radial basis function) generator. The standard RandomRBF from the river package is adapted to fit the needs of this experiment. The standard implementation produces a data stream of instances with continuous feature values modelled by a radial basis function. It generates data based on a pre-defined number of centroids spread out in the feature space. A data point is selected randomly, taking the centroids' weights into account with attributes

offset randomly from the centroid's center based on a gaussian distribution. A Random RBF is useful since a comparable implementation of this is RandomRBFDrift, an extension where concept drift can be introduced by adding a speed to certain centroids to make them move. However, this method is not used in this research. The custom RandomFeatureRBF is created to control the environment more tightly for the experiment. The only difference is that the center of the centroids is picked based on the self-defined min, max, and length of a feature value. The RFRBF will place a centroid in the middle of that feature space and generate samples from a gaussian around the center with a standard deviation. Thereby, it will only create instances coming from certain feature value areas. Each steam is initialized with the parameters defined in the customization options. The most important parameters are the mean and the standard deviation of the stream. The mean is calculated using the average of the max_featue_value - min_feature_value. The standard deviation is then applied to the gaussian used to generate the data instances.

Simulation goal: By creating a stream within a specific part of the feature space, one can simulate a group of instances that only has a certain classification. In a binary setting, this classification is negative or positive. By altering the weights during the experiment, one can increase or decrease the number of instances coming from this group.

Customization options: Seed model and seed sample for randomization. N_classes, n_features, n_centroids, class_weights, std_dev, min_feature_value, max_feature_value, and stream_name for experimental needs. An of two datastreams over time is given in Figure 3.7

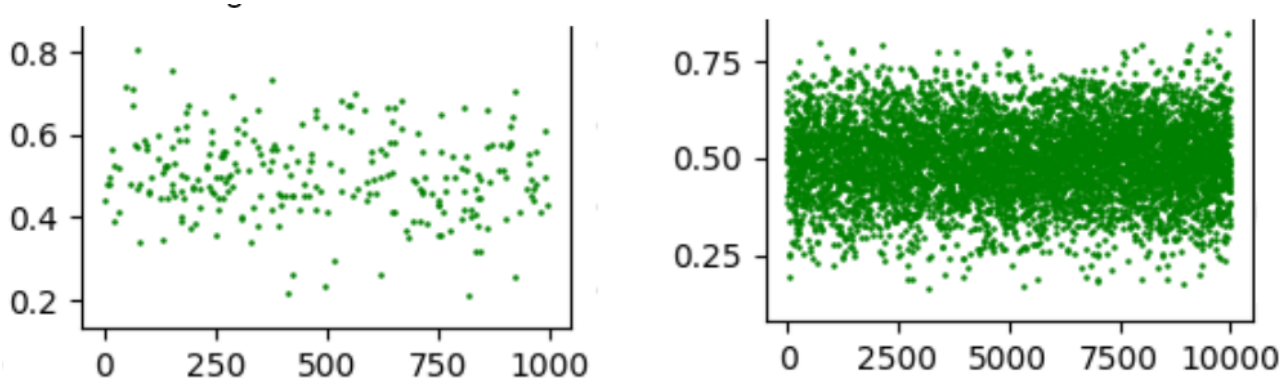


Figure 3.7: Datastream: Low max value vs high max value over time

Dataset: Prediction Influence Dataset

A data prediction influence dataset contains multiple RandomFeatureRBF datastream generators. These streams are placed evenly throughout the feature space. Each feature space is split up into predefined parts through a parameter, and 2 data streams are placed within each part: a data stream

that only produces positive instances and a data stream that only produces negative instances. The primary function of this dataset is to decide which stream gets to produce the next instance based on the weights of the streams. Whenever a true class label enters the dataset, the weights are updated based on whether the prediction was correct or incorrect. This update is done based on the prediction influence type. The strength of the update is done through the influence_multiplication parameter. In a self-fulfilling prophecy with influence_mutliplication set to 0.001, the stream that gets a correct classification gets its weight multiplied by 1+0.001 = 1.001. This self-fulfilling effect is seen in Figure 3.8 In a self-defeating experiment, this weight multiplication is done by 1-0.001 = 0.999. In a no drift experiment, no multiplication is done. It is also possible to make the drift strength linear instead of the multiplication by using the "addition" parameter value for the influence_method parameter. In addition, it is also possible to set an update delay. However, both these options are not used in this research.

Simulation goal: This dataset is used to create an indirect prediction drift in the generated samples. The conceptual effect is instances sharing information between each other, causing future instances to have modified feature values. This causes a drift in the feature values over time.

Customization options: Seed for randomizations. Influence_drift_kind and influence_multiplication for influence details.

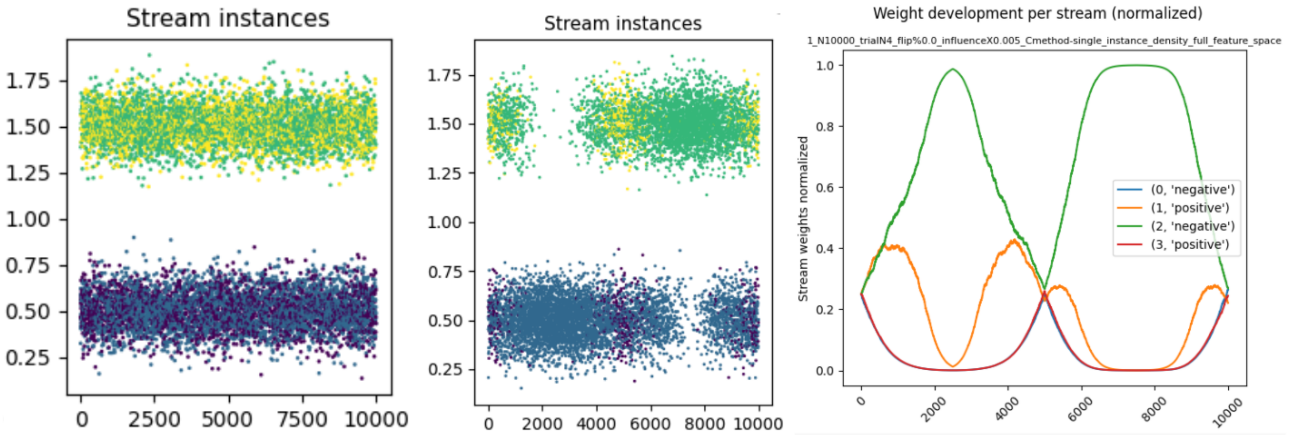


Figure 3.8: Dataset: Low influence multiplication experiment (left), high influence multiplication experiment (middle), normalized stream weight development of middle (right). Each instance colour is a different stream

Classifier: Trivial prediction pattern

A classifier is needed to predict the class of the instance. While any systematic classification model can be used in this experiment, a non-learning rule-based classifier is used. The trivial classifier pre-

dicts all instances with even feature values positive and instances with uneven feature values negative. After a predefined time period, this classification flips. Thus instances with even feature values get a negative prediction, and instances with even feature values a positive prediction. These are the regime changes that are used for the dependent variable calculations. N_trials define the number of trials, and the regime switch happens after max_samples/n_trials instances. For example, given 1000 instances and 4 trials, this switch happens after 250 instances. Implementing the A/B testing detection method is done by flipping a certain percentage of instances into the opposite classification. This creates a randomization effect which makes it possible to use average treatment effect calculations. This effect is seen in Figure 3.9

Simulation goal: Simulates an environment where a classifier is in place but where we can interfere with its classifications to do the opposite of what the classifier normally predicts.

Customization options: Experimental parameters: n_trials & flip_percentage

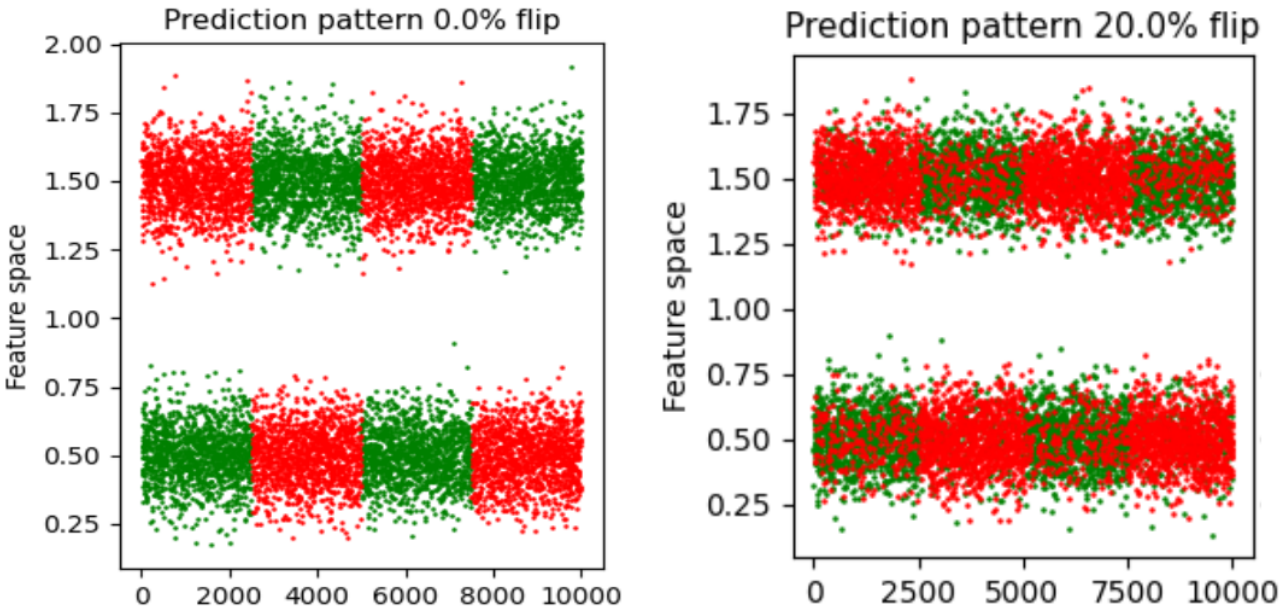


Figure 3.9: Classifier: Checkerboard prediction pattern with 0% flip (left) and 20% flip (right)

Question & Answer session

A time-ordered question and answer session simulates the environment of instances coming in over time according to the given dataset. A question is an ask to the dataset to produce an instance with features. This instance will have no classification yet. After a certain delay in time, the instance will come back in the answer part, including a ground truth of that instance.

Customization options: moment and delay for the time-space of a session. The moment is the attribute used to measure time. The delay is the amount of time to wait before revealing the ground truth

classification of the instance.

In the experimental summary plot in Figure 3.10 the interaction between the data generator and the

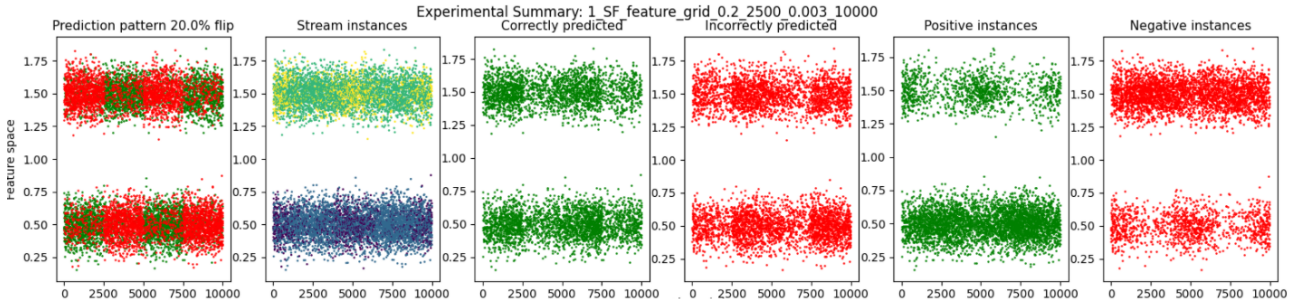


Figure 3.10: Question & Answer session: Interactions of dataset & predictions over time

classifier over time can be seen. The classifier predicts based on the prediction pattern visualized in the first graph. The second graph shows which stream the instances come from. The third and fourth graphs show which instances are correctly classified and incorrectly. The last two graphs show the true classes of the instances over time. Since the drift_multiplication is set to 0.003, which is relatively high for this experiment, the influence is easily visualized over time.

Tracking

Dataframes: Every action related to a single instance is recorded and collected in two dataframes. One dataframe is the "before_label" dataframe, and the second is the 'after_label" dataframe. The former can be used for drift detection before the ground truth arrives, thus providing a faster detection method. However, since this detection will be only based on the feature values of the instances, it will most likely be less accurate. The after_label data frame is used for the detection approach in this research.

Stream weights: The weights of each stream in the prediction influence dataset is also tracked. These are primarily used as a visual aid in the analysis of the experiment.

Metric: A metric is used to track how good the predictions are that the model makes. Since the classifier used in this research is a trivial one, this metric is only used to validate the experimental influence type. Self-fulfilling experiments should see an increase in the performance metric over time, and self-defeating experiments should decrease. Any metric that fits the model is possible to use.

3.1.2 Drift detection setup

The drift analysis is done separately from the experiment. This is currently done on an ex-post basis, which means it is only analyzed after the experiment is finished. This can, however, be transformed to be done after each trial or in-real time. It consists of the following components:

Spacetime fragmentation: To calculate distributional changes, the start and end time of a trial need to be defined. Based on these parameters, the time-space is divided into n_trial blocks. Based on the change_calculation_method parameter, that timespace can be further divided into feature spacetime cells (like the prediction pattern). These need to be defined to narrow down which part of the feature space is responsible for the prediction drift.

Customization options: feature_space_length, feature_space_split, and time_space_split are all used for the spacetime fragmentation. Time_space_length determines the trial length. This is useful for determining how long it takes for the influential effect to take place.

Trial distribution change calculation: For each class in the experiment, the instances are gathered and the distribution changes calculated. These calculations are done within the defined spacetime setup. It is chosen to split up each trial by 4 time parts. The first part is used for the trial start, and the last part is used for the trial end. For each of these two parts, a Kernel Density Estimation function is fit on the feature values of the instances.

Kernel Density Estimation: This is a neighbour-based approach to visualize the data distribution by fitting a non-parametric model to all the data points. By using a gaussian KDE, a gaussian curve per instance feature value is contributed to the total which results in a smooth density estimate derived from the data. 2 KDE's are created: The first uses the trial's start instances (t=0), and the second uses the end of the trial (t=1). The instances of t=0 get scored on both KDE's. This produces the log-likelihood of each sample under each model. The exponent of these log-likelihoods is taken to produce the density of the point in each timeframe. These densities are then used to check the difference of density estimations between the different timeframes in a trial. All these KDE's can be seen in Figure 3.11. Two calculation methods are created for this method. The first one uses only the trials' instances to calculate fit the KDE. This is called the single_instance_partial_feature_space calculation method. The second method uses the full feature space per timeframe. Thus, if 2 trials are in the same timeframe, all instances of these 2 trials will be used for the density calculation. This method is called single_instance_full_feature_space. Ideally, for each instance, a different KDE should be fit

on all trial instances - the specific instance. However, since this method uses a large sample size, this is too resource-intensive. The decision was made to use a simpler method. Each KDE that is fit, is denormalized, the peak standard normal value is subtracted, and then the KDE is normalized again by the amount of instances of the start of the trial and the end of the trial - 1. A picture of the corrected KDE's per trial can be seen in Figure 3.12

$$Corrected\ KDE: \ (KDE_{t=0} * n_{t=0} - PDF(0))/n_{t=0} - 1 + n_{t=1}$$

This corrected KDE prevents the number of instances influencing the density estimation. Thus, it is kind of invariant to intrinsic drift. An integration is done per KDE to validate if the KDEs are calculated correctly. Each integration should result in 1. This is the case for every KDE that is fit on the data in this research.

Customization options: the estimator and the bandwidth can be changed for the KDE. However, the bandwidth is chosen to be 0.1, and the estimator does not matter if enough data is used. Thus, the "gaussian" is chosen. Change calculation method: single_instance_density_full_feature_space and single_instance_density_partial_feature_space

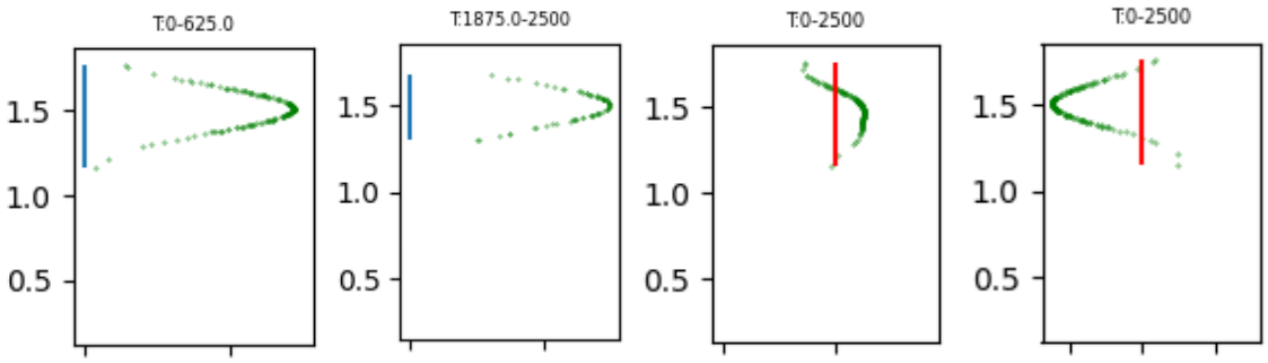


Figure 3.11: KDE: Start, end, difference, and corrected KDE for trial 1, feature values 1:2

Hypothesis testing: The correct and incorrect relative density changes above are then used for a significance test. The testing is done using the Mann Whitney U test. This is a non-parametric version of the independent samples t-Test. The test checks if there is a rank-sum difference between the two groups without assuming any data distribution. The test assumptions are: X and Y are independent random samples with at least ordinally scaled characteristics, and X and Y should be of the same shape. This test is chosen because it makes no assumption on the distribution of the data, and the sample sizes can be different from each other.

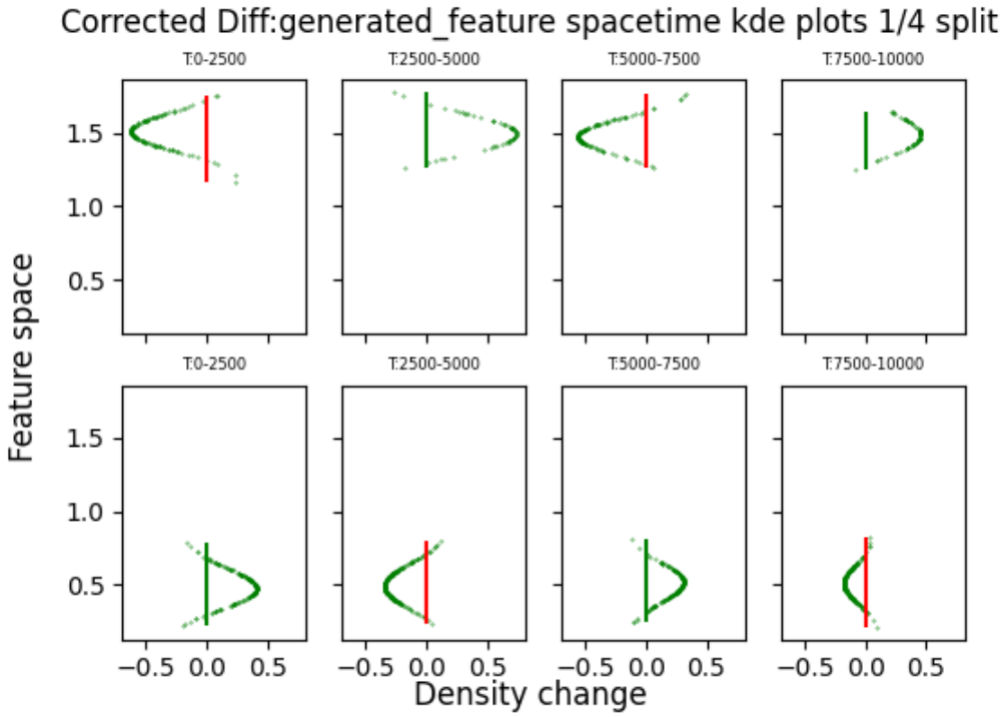


Figure 3.12: KDE: The corrected density changes per feature trial, with classification regime as vertical line

Mann Whitney U test application in method:

- Treatment / Independent variable: Positive classification / negative classification
- Dependent variable: relative Kernel Density % Change from T=0 to T=1 within a trial.
- H0: There is no difference between the relative density % changes ranks of classification treatments
- H1: There is a difference between the relative density % changes ranks of classification treatments

The implementation of the test is done using Scipy's stats package. Alpha is the probability of getting a sufficiently low value for the test statistic by chance under the null hypothesis. Significant: This probability is less than alpha. Insignificant: This probability is more than alpha. In this test, an alpha of 0.01 is used instead of 0.05. This is done to ensure that the samples are significantly different from each other. If the p-value of the statistical test is lower than alpha, the H0 is rejected. Thus the difference is significant. Also, continuity correction is applied in the Scipy Mann-Whitney U test. This is done since the used variables are on a continuous scale. It is assumed that there is no need to take care of multiple hypothesis testing using the Bonferroni correction method since this significance test

is only performed twice per experiment.

Post-processing of influence detection: A drift detection is fired when a hypothesis test is deemed significant. This means the correctly classified instance changes are significantly different from the incorrectly classified ones. However, now the drift needs to be classified, quantified, and timed. This is done in the post-processing of the significance test.

The drift quantification: Per true class, the drift quantification is done by dividing the correctly classified densities and the incorrectly classified densities. The median of these density changes is the average treatment effect relative to trial time (max_samples/_trials).

The drift classification is done using the quantification per class using the correct vs incorrect classifications. If it is a correct classification and the average treatment effect > 0, this true class has a self-fulfilling drift, increasing these instances. If it is a correct classification and the average treatment effect < 0, then this class has a self-defeating drift which decreases this class. Vice versa, if the classifications are incorrect for the class, and the average treatment effect > 0, then it has a self-defeating drift which increases this class. If the average treatment effect is < 0, then it has a self-fulfilling drift, decreasing this class.

Evaluation method setup

Sensitivity analysis

Single run analysis: A single full experimental cycle will be run with the predefined parameters. Afterwards, a conclusion will be given about the influences apparent in the experimental setup. This can be a minimum of 0 influences and a maximum of 4 influences since there can be a difference in 2 prediction drift influence types, self-fulfilling and self-defeating influence, and 2 different classes of instances being affected by the influence, the correctly and the incorrectly classified. A validation step can determine if the influence detection prediction was correct or incorrect. Run n time analysis: The same cycle is done as in the single run analysis, only an N amount of times with the same parameters. Because there are various randomizations possible in the setup of the environment, this function can determine how many comparable experiments are predicted in the desired way. The more randomization is added to the environment, the more volatile the results.

Influence type search: Run same hyper-parameter set per influence type, n_times. This analysis approach is useful when comparing a specific parameter set with each prediction drift type. The output

of this experiment depends on the n_times parameter. If this is lower than 4, then a stacked boxplot graph is created for the outputs of the experiments. If it is => 4, then the graph becomes too big and too complex to visualize. Then, an aggregated boxplots chart is created. This uses the median of each experiment and aggregates those per prediction influence type in a single boxplot. The output size of this analysis is: number of runs per parameter set * number of influence types.

Grid search: The grid search is the most important analysis method of this research. It will run the same hyper-parameter set, with 1 parameter changing every run, n_times. This is useful when a comparison is needed of identical environments except for 1 parameter changing each time. This is done according to a predefined set of single parameter values. For example, the influence_multiplication grid list can contain the following values that will be compared: [0.0001, 0.0005, 0.001, 0.005, 0.1]. This will create an output of size 5 * 100 = 500 if the n_runs is set to 100.

Using these methods, answers can be given to questions like: In which environments does the detection method perform well? At which breakpoints does the performance degrade? Which variables create an exceptional environment?

Performance metric calculation All classified influences are aggregate per experiment to calculate the performance metrics. SF & SD give 2 influences per true class of instances. Thereby, the experiment produces 4 influences. If all 4 influences are classified correctly, the experiment is correct. If 3, 2, or 1 influences are classified correctly, the experiment is partially correct. If 0 influences are classified correctly, the experiment is incorrect. For None experiments, the significance test should reject the H0 hypothesis. This means that each true class will have 1 influence, "None". This totals to 2 influences per experiment. If only 1 of the two influences is classified as None, the experiment is partially correct. With these aggregated experiment performance numbers, the following metrics are calculated:

$$Accuracy: \quad n\_correct\_experiments \; / \; n\_total\_experiments * 100$$

$$False - negative\ rate: \ (for\ SD\ and\ SF) : 1 - Accuracy$$

$$False - positive\ rate: \ (for\ None) : 1 - Accuracy$$

Characteristics of method:

Pros of method:

Drift detection: The detection approach will detect a drift if the ranked relative distribution changes of the correctly classified instances vs incorrectly classified instances will be significantly different.

Drift classification: By post-processing the statistical result per true class instances, it can be checked what kind of direction the detected drift has.

Drift quantification: The global average treatment effect can quantify drift that affects the population as a whole. This is, however, not one-to-one applicable to individual instances. The main reason for this limitation is the network effect designed into the experiment. Through this effect, conceptually, the instances are influenced by other instances over time by sharing the prediction knowledge. This, in turn, skews the individual treatment effect (ITE), making it unreliable.

Drift localization: By applying a slight local randomization approach to the generated data, a local average treatment effect could be calculated without disturbing the regular operation of the classifier. However, this is not implemented yet.

Self-made control group: The randomization is done through a checkerboard classification pattern with a self-made control group created by flipping the classifications of a certain percent of instances. Prediction influence drift can be measured in the checkerboard cell using this control group.

Counterfactuals: Such a method could answer counterfactual questions like "How would the system behave if we would have replaced the classifier?" and "How would future data points behave if we gave the current instances different classifications?"

Modular experiment: Each experiment component can relatively easily be replaced by a comparable component with the same in and outputs. For example, another calculation method or another significance test is easily interchangeable since a lack of interdependency.

Validation: Each experiment can be made deterministic or stochastic for validity testing by changing the seed of the data generator and the seed of the stream choosing.

Networked treatment diffusion: Network treatment effects are designed into the experiment. In a self-fulfilling prophecy, the effects are simulated of instances that get classified correctly "spreading" this positive information, and thereby more similar instances will use the classifier.

KDE correction: This method uses a correction of KDEs based on the number of instances across the feature space per trial. Because of this, artifacts are removed that are created by instances moving from one feature space to the other. Therefore, making the calculations invariant to the absolute change of the instances.

Distribution agnostic: The detection approach can be used on any numerical dataset provided by not assuming any distribution of the changing data because of the Mann Whitney U test.

Model agnostic: For this method, it does not matter what the model is, how it learns, or how it behaves. All that this method needs are the to be classified instances and their feature values. Using these, it can create a randomization pattern to apply to the classifications scheme through classification regimes. Thus, the model can be a deep learning classifier or a simple systematic non-machine learning decision engine.

Limitations of method:

No individual treatment effect: The method cannot give conclusions about the influence quantification on a specific single instance (ITE). This is because of the network effect by design in influencing experiments which make the ITE calculation biased. However, a sub-population level influence drift quantification can be given.

Treatment interference: The method is created on the assumption that there are causal treatment interference effects through in-group knowledge transfer of classification outcomes An (2018).

Ex-post analysis: The detection is done on an ex-post basis in the current form. This is done according to the real-world practice of an algorithm needing an audit. However, this detection approach can also be used on a semi-real-time basis with streaming data by applying a rolling window.

Significant data need: The experiment can only detect drift if instances with their features, classifications, and true labels are fed to the detector. Thus, it can only make drift conclusions with ground truth available. This makes it hard to implement in a real-time real-world setting since the ground truth is often hard to get and hardly ever arrives in-real time.

Large concentrated sample: For the significance test to be reliable, a large enough sample of instances is needed, and these instances should not be spread out far from each other in the feature space.

Single cause: Confounders are not known apriori and thus not considered. The drift detection approach only considers its predictions and makes conclusions about data change in a future time.

Global conclusion: The current method can only give a global conclusion about whether there is influence drift, its quantification, and its direction in the experiment. It cannot yet zoom into the conclusion and localize the drift.

Table 3.3: Method differences Jelsema et al. (2021) vs current method

Research differences	(Jelsma et al, 2021)	(Dragomiretskiy et al, 2021)	Explanation
Goal	Detect and classify prediction influence drift	Detect, classify, and quantify prediction influence drift	For how many instances the experiment will last.
Data streams	RandomRBF	RandomFeatureRBF	By how many flip trials the max_samples should be split.
Dataset	PredictionInfluence (n streams)	PredictionInfluence (2*n_partial_feature_space streams)	The percentage amount of instances predictions to flip
Classifier	Simple learning model	Trivial classification pattern	What kind of influence classifications create on the environment.
Detection method	KDE confusion matrix trajectories with linear regression model projection	KDE relative change calculation per trial of A/B testing checkerboard prediction pattern, and flip% for randomization	The strength of the influence drift.
Treatment	True labels of instances	Predicted labels of instances	Different method options for the within trial change calculations.
Drift assumptions	Intrinsic drift is gradual and no concept drift	No intrinsic drift and no concept drift	The method for post-processing a significant statistical test.

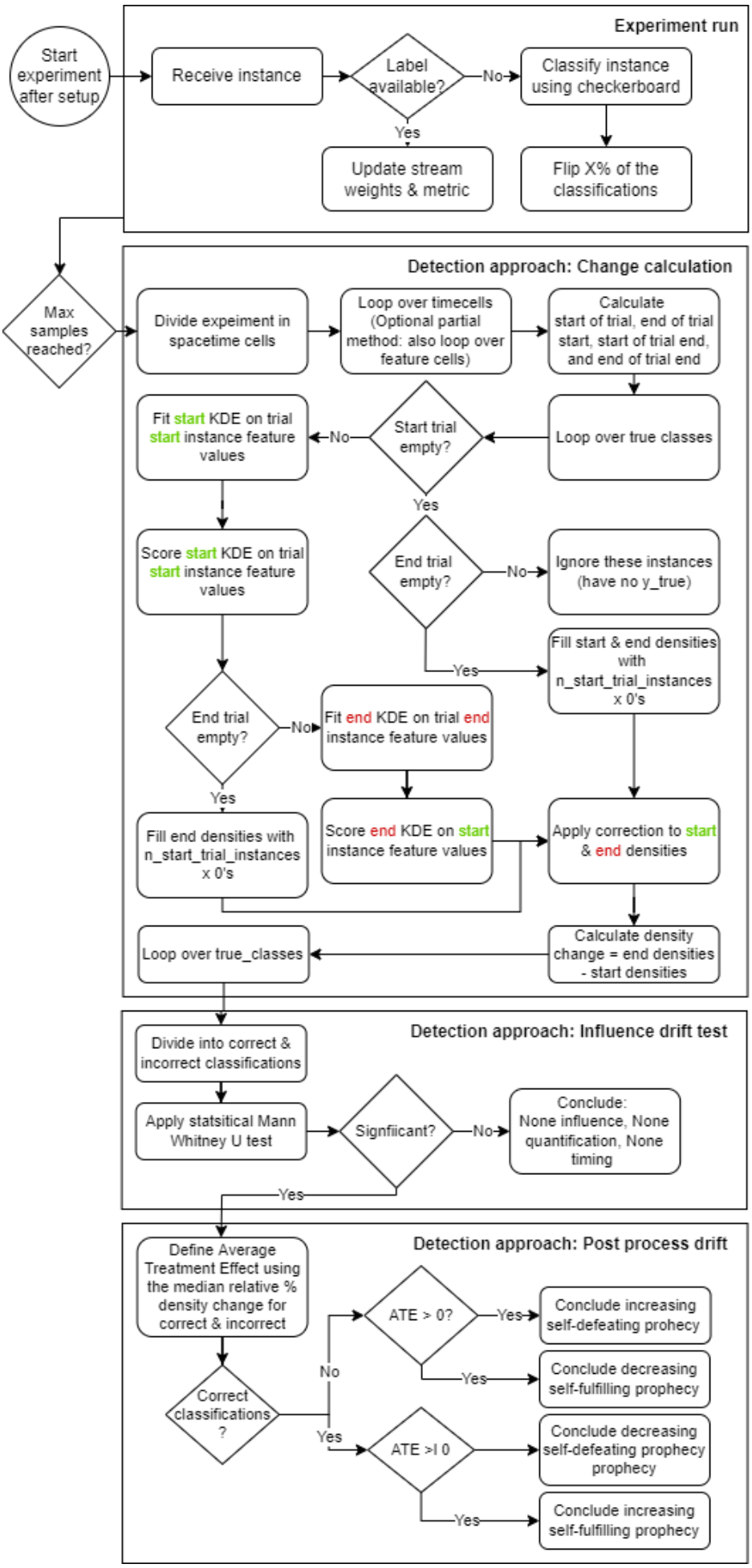


Figure 3.13: End-to-end influence prediction detection method

Chapter 4

Experiment and Results

In this chapter, a single run experiment will be done and the plots and results that it produces will be shared. The fixed parameters of the run can be found in Table 3.1. The variable parameters from Table 3.2 are filled in, in Table 4.1. The parameters are chosen to try to mimic an experiment where the drift is not detectable visibly through the experiments, but still big enough to be detected by the detector. This will be done for a self-fulfilling case and for a none drift experiment.

4.0.1 Self-fulfulling drift experiment

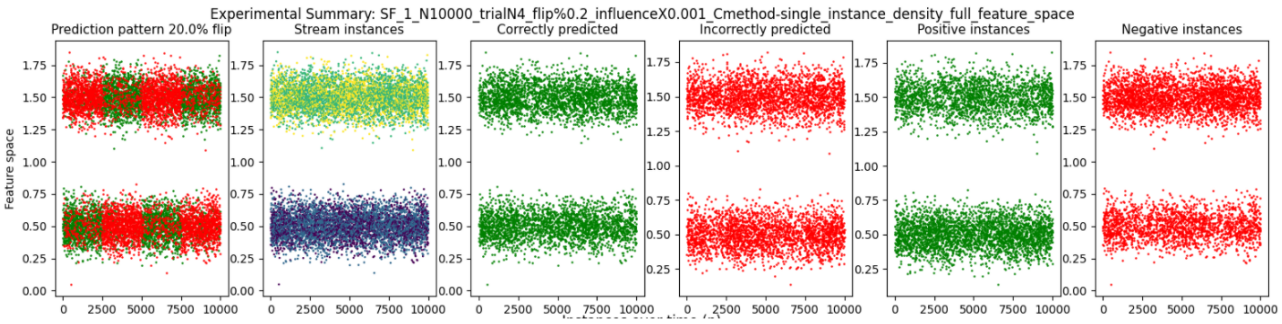


Figure 4.1: Experimental summary: Visually, there is no significant drift to be seen in these plots

Stream weights: In Figure 4.2, the orange stream produces positive values, the green stream produces negative values. Weights for both the streams go up in trial 1. Most likely, The orange stream is producing instances with feature values 0-1 since the prediction pattern is green here (left

Table 4.1: Variable parameter values for the single run experiment

Max_samples	n_trials	Flip %	Method	Statistical test	Alpha
10.000	4	20%	Full_feature	Mann Whitney U	0.01

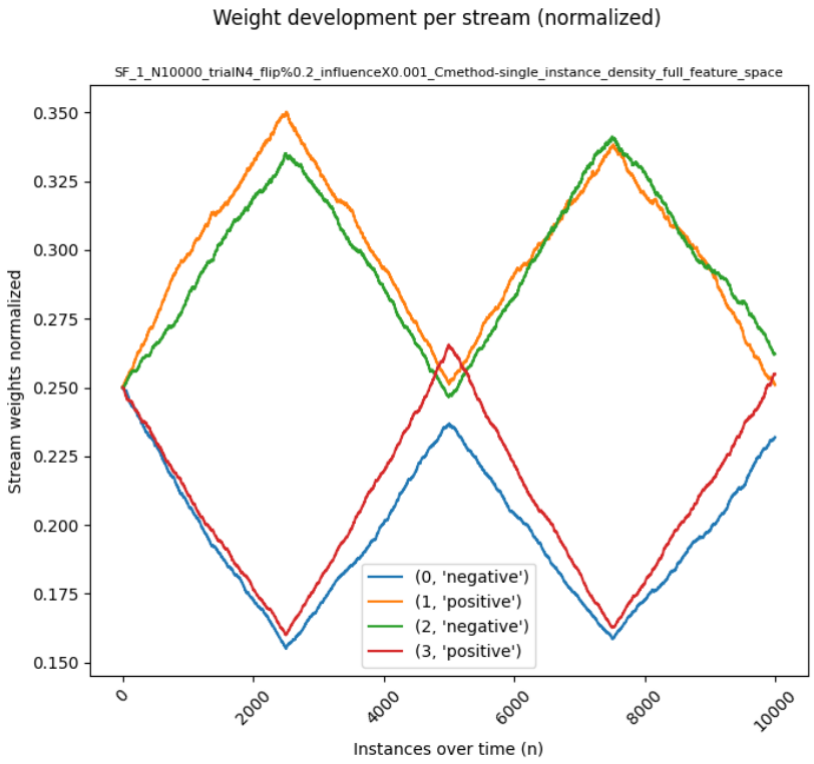


Figure 4.2: Normalized stream weights development

bottom of Prediction pattern plot) and the green stream is producing feature values between 1-2, since the prediction pattern is red here (left top of Prediction pattern plot).

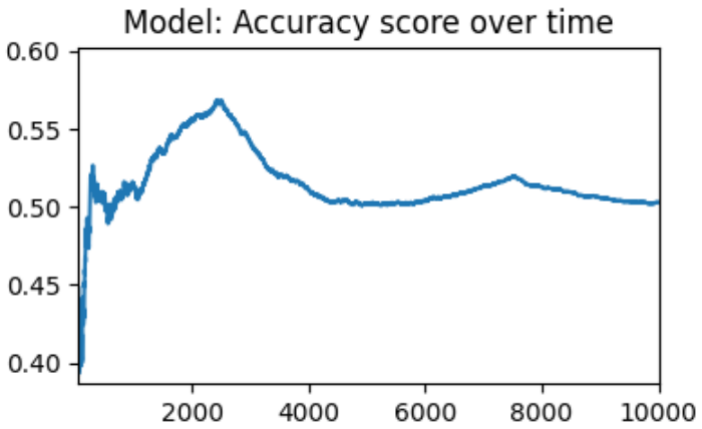


Figure 4.3: Model performance metric over time: Accuracy

Metric: The metric of the trivial model goes up in the first trial (from 0 till 2500) as seen in Figure 4.3. The model performance drops when the regime change happens since it is still classifying instances according to the old regime. It needs time till the instances react to this change, and the model performance will go up again because of the self-fulfilling prediction influence drift.

KDE's: Gaussian KDE fits on the first part of the trial. Since max_samples = 10.000, and n_trials = 4, which makes 2500 instances per trial. The trial is divided into 4 parts, so the first part goes from

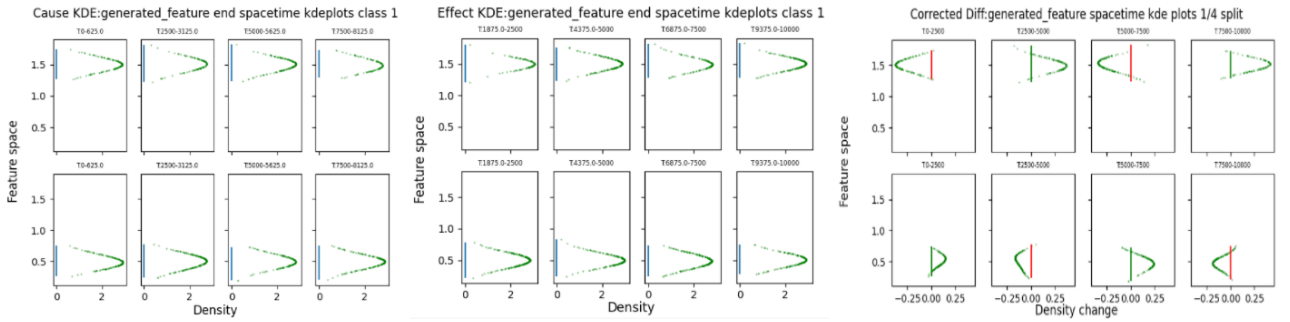


Figure 4.4: Experimental summary: Visually, there is no significant drift to be seen in these plots

0 instances till 2500 / 4 = 625. The same fits are done for instances of class 0. Ending KDE plots per spacetime cell for class 1

Gaussian KDE fits on the first part of the trial. The last part of the trial goes from ¾ instances till the end of trial. This is in this case from 1875 till 2500. The same fits are done for instances of class 0. Corrected KDE plots per spacetime cell for class 1

The 0 line is coloured based on the regime in that spacetime cell. Thus, there is a negative regime going on on the top left. The corrected densities are below 0. Thus this hints at a self-fulfilling prophecy. The same calculations are done for instances of class 0.

Signficiance conclusion: The corrected density changes over all trials for correctly classified instances and incorrectly classified instances are used for the statistical Mann Whitney U test. The result of class 1 can be found in the code output of Figure 4.5 and class 0 in Figure 4.6

```
Statistical test for class 1
N incorrect instances 690
N correct instances 600
ΔN 90
Method: classification True_Class: 1 MannwhitneyuResult(statistic=74760.0, pvalue=1.1089582571486352e-87) N: 1290
1.1089582571486352e-87 < 0.01 test is significant
SIGNIFICANT influential effect of classifications on behaviour of class 1 instances
```

Figure 4.5: Significance test class 1

```
Statistical test for class 0
N incorrect instances 672
N correct instances 538
ΔN 134
Method: classification True_Class: 0 MannwhitneyuResult(statistic=80004.0, pvalue=8.8589666616255846e-63) N: 1210
8.8589666616255846e-63 < 0.01 test is significant
SIGNIFICANT influential effect of classifications on behaviour of class 0 instances
```

Figure 4.6: Significance test class 0

Drift type classification: Since both statistical tests reject H0, a drift is detected in both classes.

The relative density change percentages are post-processed to classify the drift types. This is done according to the following boxplots in Figure 4.7.

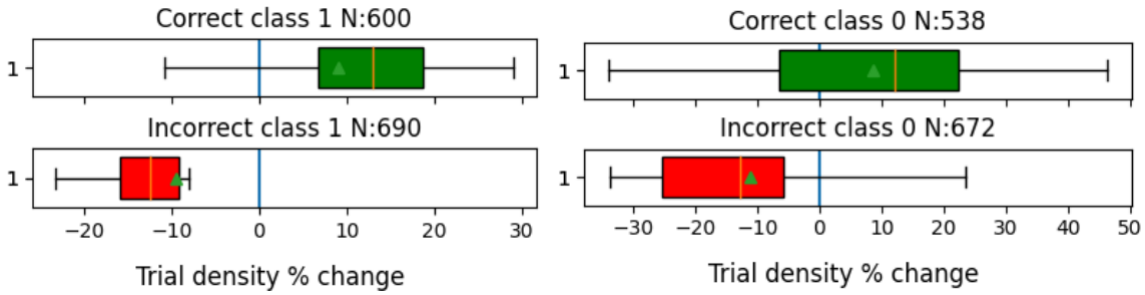


Figure 4.7: Drift type classification: density distributions of both classes

Both class boxplots have the same conclusion: The median of the density % changes instances is positive when they are classified correctly, and the median is negative when the instances are classified incorrectly.

Drift quantification The average treatment effect per class population is calculated using the median density % change. This results in the drift conclusion for class 1 in 4.8 and class 0 in 4.9:

```
classification ex-post analysis
correct classifications increase class 1 densities with an avg of 9 % ( N: 600 std: 13.438708326434115 ) , after 2500 time: SF prophecy in class 1
incorrect classifications decrease class 1 densities with an avg of -10 % ( N: 690 std: 11.129075699391915 ) , after 2500 time: SF prophecy in class 1
```

Figure 4.8: Drift quantification: average treatment effect code output

```
classification ex-post analysis
correct classifications increase class 0 densities with an avg of 9 % ( N: 538 std: 18.369665699621244 ) , after 2500 time: SF prophecy in class 0
incorrect classifications decrease class 0 densities with an avg of -11 % ( N: 672 std: 15.551764547166606 ) , after 2500 time: SF prophecy in class 0
```

Figure 4.9: Drift quantification: average treatment effect code output

Conclusion: The self-fulfilling effect is classified correctly.

4.0.2 None drift experiment

Experimental setup: In an one experiment, the model accuracy and stream weights stay flat throughout the experiment. This can be seen in Figure 4.10.

Significance test: The significance tests for both classes rejects the H0. Thus no drift is found in the distributions of the density % changes in Figure 4.11

Conclusion: The none effect is classified correctly.

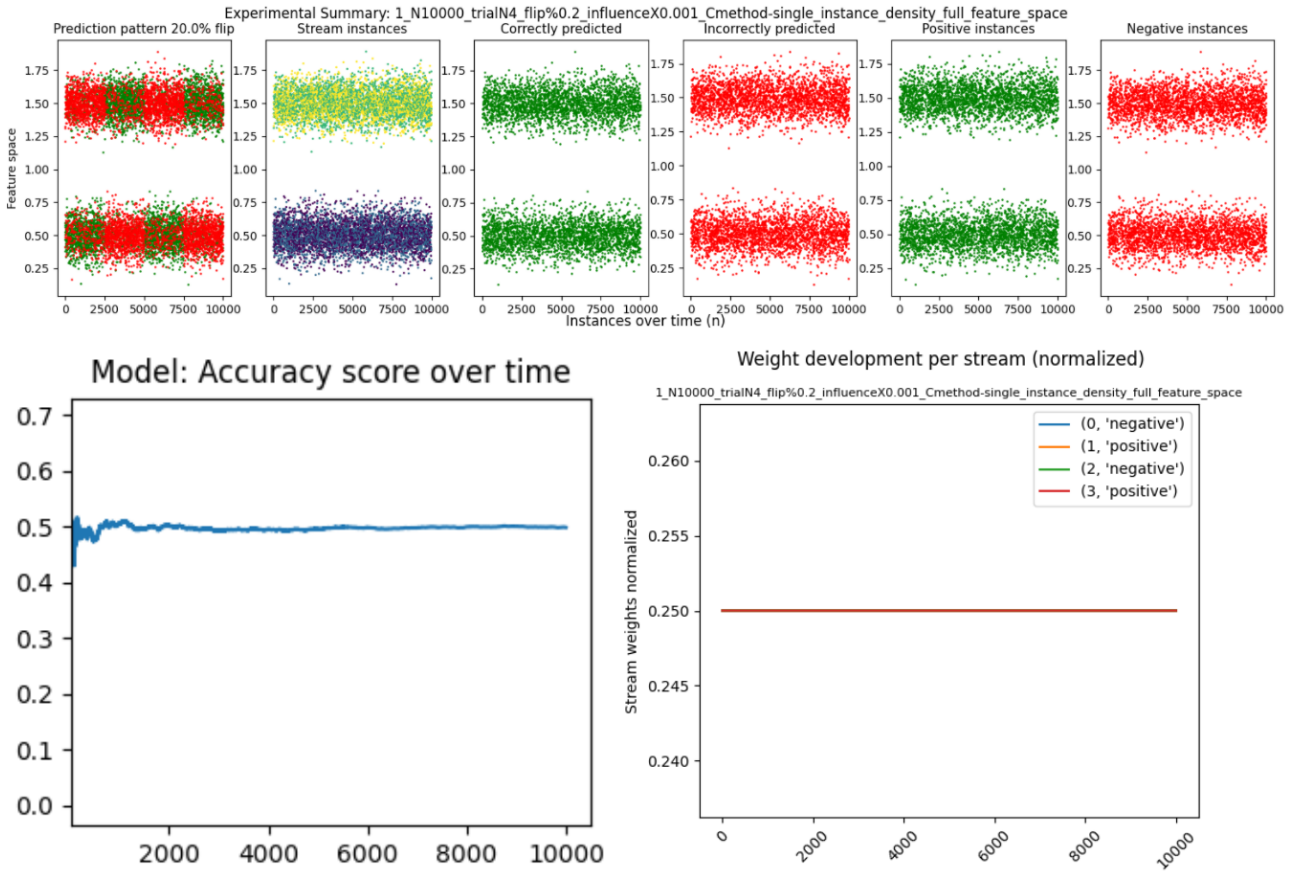
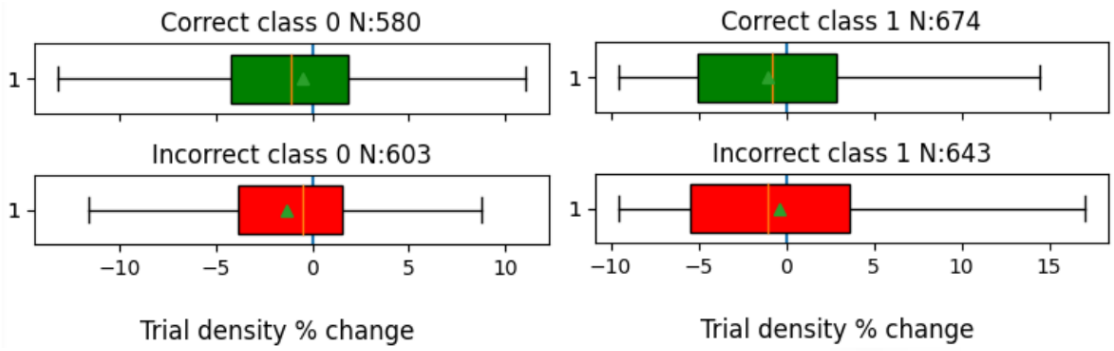


Figure 4.10: Experimental summary: None experiment




```
Statistical test for class 0
N incorrect instances 603
N correct instances 580
ΔN 23
Method: classification True_Class: 0 MannwhitneyuResult(statistic=173820.0, pvalue=0.42906936804812823) N
0.42906936804812823 > 0.01 test is insignificant
NO SIGNIFICANT influential effect of classifications on behaviour of class 0 instances
```

```
Statistical test for class 1
N incorrect instances 643
N correct instances 674
ΔN -31
Method: classification True_Class: 1 MannwhitneyuResult(statistic=208336.0, pvalue=0.11294781228982859) N
0.11294781228982859 > 0.01 test is insignificant
NO SIGNIFICANT influential effect of classifications on behaviour of class 1 instances
```

Figure 4.11: Significance test: None experiment

Chapter 5

Method Evaluation

In the last chapter, two single run experiments were done to validate if the detector can classify prediction influence drifts. However, these are done only once, so results may vary through runs. Therefore, an evaluation will be done by running these experiments hundreds of times. There are two meta-experiment methods available. One method aggregates the results per drift influence type. The other one is a grid search method which aggregates the results based on the feature value it is testing. Both methods will be used in this section to give more detailed answers of the inner workings of the method.

5.0.1 Multiple experiments per influence type

Single setting comparison: To show the differences of each influence type, this test is run with the same parameter settings. These can be found in Table 5.1. This means that for every drift type, 100 experiments are done. Thus, the boxplot visual aggregates the median KDE % changes of 300 experiments with a total of 1150 influences. This boxplot can be seen in Figure 5.1.

The conclusions here are clear. The None experiments calculate drift strength values around 0, with a variance of -5% and +5%, which is the intrinsic drift of the randomness of the experiment. Self-fulfilling experiments increase the densities of correctly classified and decrease the densities of incorrectly classified instances. Vice versa, self-defeating experiments decrease the densities of correctly classified instances and increase the incorrectly classified ones. These drifts' strengths are around -12% and +12% based on the median values.

The result in Table 5.2 shows that all the self-fulfilling and self-defeating experiments are clas-

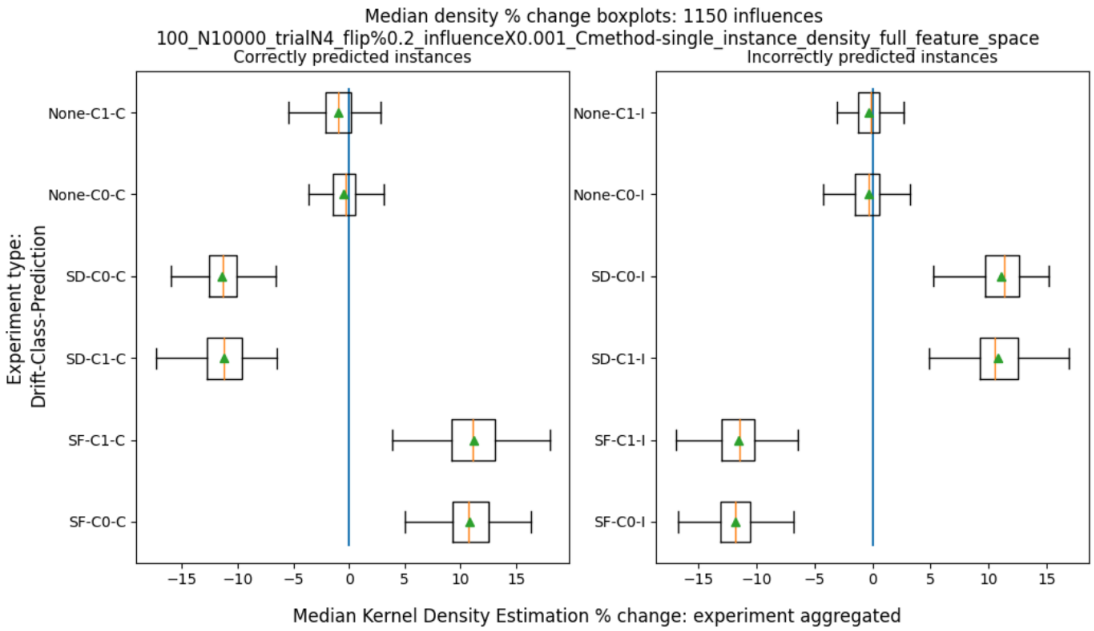


Figure 5.1: Model performance metric over time: Accuracy

Table 5.1: Variable parameter settings for the single run experiment

n_times	**Max_samples**	**n_trials**	**Flip %**	**Influence X**	**Method**	**Statistical test**	**Alpha**
100	10.000	4	20%	0.001	Full_feature	Mann Whitney U	0.01

sified correctly. This shows that the drift detection is robust and does not change based on the randomness of the experiment. However, the None experiments only get correctly classified partially. This means that only the positive class or the negative class is classified as having a None drift per experiment. This looks unsatisfactory. However, the reason and the solution for this result will be discussed in the flip % parameter grid search section below.

Calculation methods: Single instance partial feature space vs single instance full feature space. The detection method makes use of 2 different calculation functions. One uses the full feature space to calculate the density change per single instance, and the other uses the partial feature space which is split by predefined parameters. The methods are compared through the use of two graphs in Figure 5.2. The partial method is on the left and the full method is on the right. The parameter settings can be found in Table 5.3

Table 5.2: Resulting table of the multi-influence experiment

influence_drift_type	n_correct_experiments	n_partial_correct_experiments	n_incorrect_experiments	accuracy	false_negative_rate	false_positive_rate
SF	100	0	0	100.0%	0.0%	0%
SD	100	0	0	100.0%	0.0%	0%
None	2	46	52	25.0%	0%	75.0%

Table 5.3: Variable parameter settings for calculation method comparison

n_times	Max_samples	n_trials	Flip %	Influence X	Statistical test	Alpha
10	10.000	4	20%	0.001	Mann Whitney U	0.01

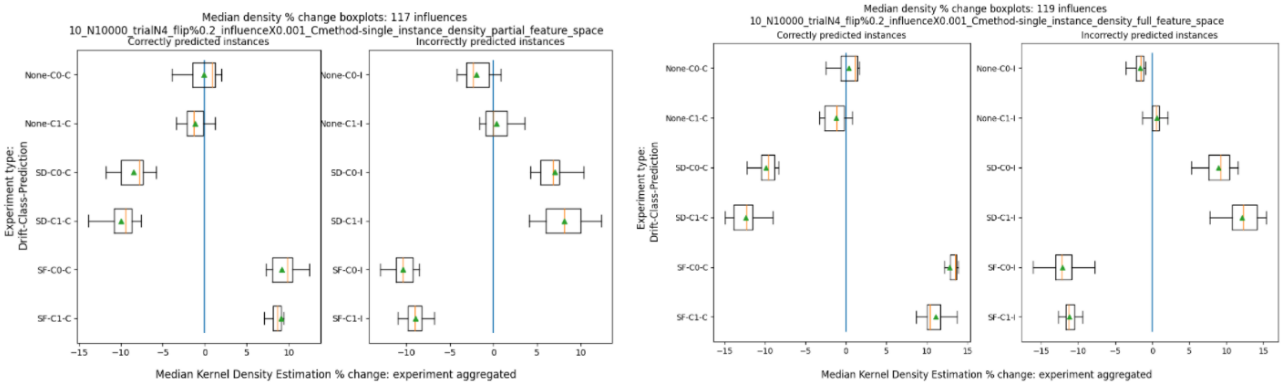


Figure 5.2: Calculation method differences: Partial feature space (left) Full feature space (right)

Result interpretation: Both methods perform well at classifying drift as seen in Table 5.5 and Table 5.4. However, the change calculations of the full method seem to give more drastic drift conclusions. The resulting table shows that the full method has a higher false-positive rate for None experiments. This happens because the change calculations that are used produce more significant results based on the Mann Whitney U statistical test since the sample sizes of the full method are larger than the partial method. It is known that the signed ranked test is sensitive to large sample sizes. In these situations, it is prone to produce Type 1 errors (Zimmerman, 2010). Thus, more significant drifts are concluded, which are low in drift strength (<5%). The full method should be more robust since it uses all the instances of a time block. In a scenario where all the instances of one feature part disappear, the full method will create fewer artifacts because the denominator is the number of all instances in that time frame instead of a 0 at the end of a partial feature space.

In scenarios where the instances are divided equally across the feature space, both methods perform equally well as seen in the plots.

Conclusion: Given these results and the theoretical reasons behind each method, the full feature space method is used in the following evaluation methods.

Table 5.4: Table result: Partial feature space method

influence_drift_type	n_correct_experiments	n_partial_correct_experiments	n_incorrect_experiments	accuracy	false_negative_rate	false_positive_rate
SF	10	0	0	100.0%	0.0%	0%
SD	10	0	0	100.0%	0.0%	0%
None	0	3	7	15.0%	0%	85.0%

Table 5.5: Table result: Full feature space method

influence_drift_type	n_correct_experiments	n_partial_correct_experiments	n_incorrect_experiments	accuracy	false_negative_rate	false_positive_rate
SF	10	0	0	100.0%	0.0%	0%
SD	10	0	0	100.0%	0.0%	0%
None	0	1	9	5.0%	0%	95.0%

Table 5.6: Variable parameter values 50% flip validation

n_times	Max_samples	n_trials	Flip %	Method	Statistical test	Alpha
10	10.000	4	50%	Full_feature	Mann Whitney U	0.01

5.0.2 Multiple experiments per grid search

Flip % change: These graphs answer how the randomization of the instances influences the experiment's outcome. Through the setup of the randomization pattern, 50% of flipped instances make the experiment completely random. A 0% flipped percentage will classify all instances according to the defined checkerboard pattern. If 100% of the instances are flipped, classifications will be made on the inverse of the predefined checkerboard pattern. Thus, when an experiment is started with self-fulfilling influential drift, flipping more than 50% of the instances will influence the instances in a self-defeating way. Thus the most relevant values for the flip % sensitivity check are from 0% till 50%. Given this mechanic, it can be said that when 49% of instances are flipped, 1% of the instances carry influential power in the experiment.

50% flip instances drift type comparison. Another interesting thing to look at is how the 50% flip percentage influences all drift types. With a 50% flip, all the classification of the instances is done randomly. This is visualized in Figure 5.3 with influenceX of 0.001 on the left graph, and with 5x that influenceX with influenceX of 0.005 in the right graph. These experiments are done with the parameter setting of Table 5.6.

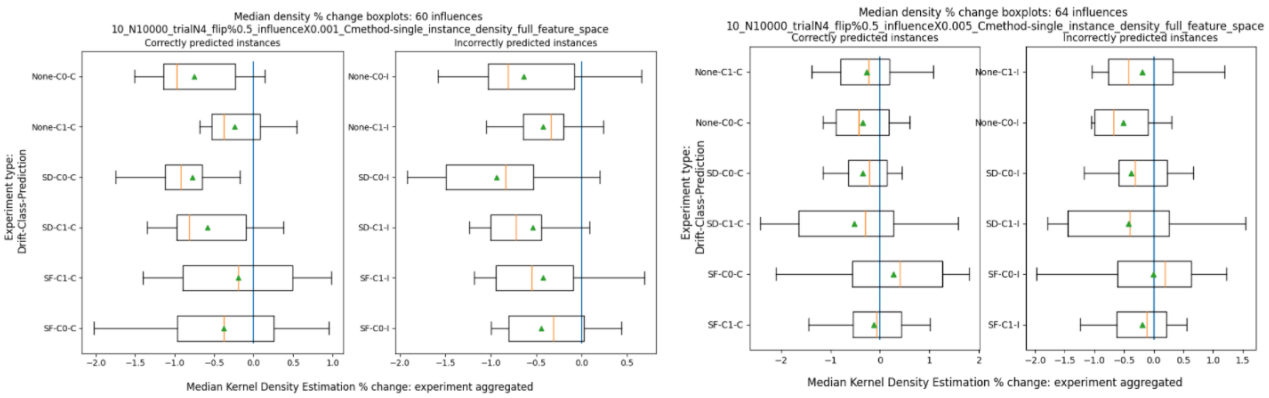


Figure 5.3: Flip% validation: 0.001 InfluenceX (left), 0.005 InfluenceX (right)

Interpretation: When 50% of the instances are flipped, every experiment, with whichever drift it is initialized, turns to become a None influential experiment. The outcomes are not dependent on the influenceX parameter as the 2 above plots are compared to each other.

Drift influence type compared to flip%

Grid search of flip% on SF and SD drift experiments

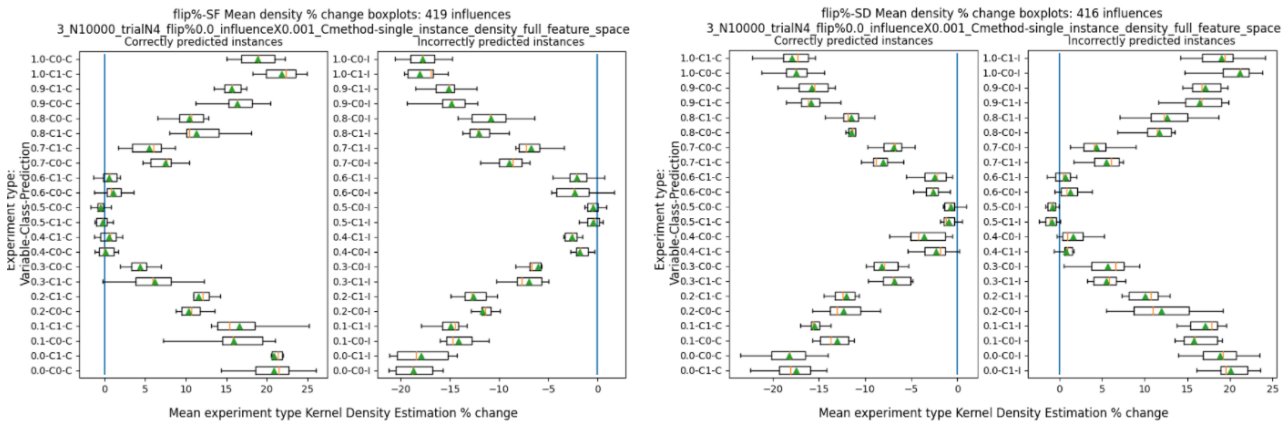


Figure 5.4: Flip% validation: Self-fulfilling drift (left), self-defeating drift (right)

Interpretation: Looking at Figure 5.4, when 0% of instances are flipped, the self-fulfilling effects are the largest. Increasing this percentage till 50% makes the drift quantity less. When the flip percentage is set to above 50%, then the drift becomes a self-defeating prophecy, since the correctly.

Grid search of flip% on None drift experiments

Interpretation: The plot from Figure 5.5 shows that when 50% of the instances are flipped, the detection of no drift works well. Given the same parameter set as the SF and the SD experiments, the median % change stays below 5% for all flip percentages. This makes it safe to assume that up until 5% density change, the drift is created by the deterministic nature of the experiment setup.

The Wilcoxon-Mann Whitney U statistical test is an oversensitive test when using it on large sample sizes (Zimmerman, 2021). These scenarios are biased for rejecting H0, which gives a high false-positive rate for None experiments. Since the experiments were done with this approach range from 1k to 100k samples, this test spots the +-5% as significant density change. Thus it classifies most of the experiments as a prophecy. A threshold of 5% median density change percentage can be used to tackle this sensitivity.

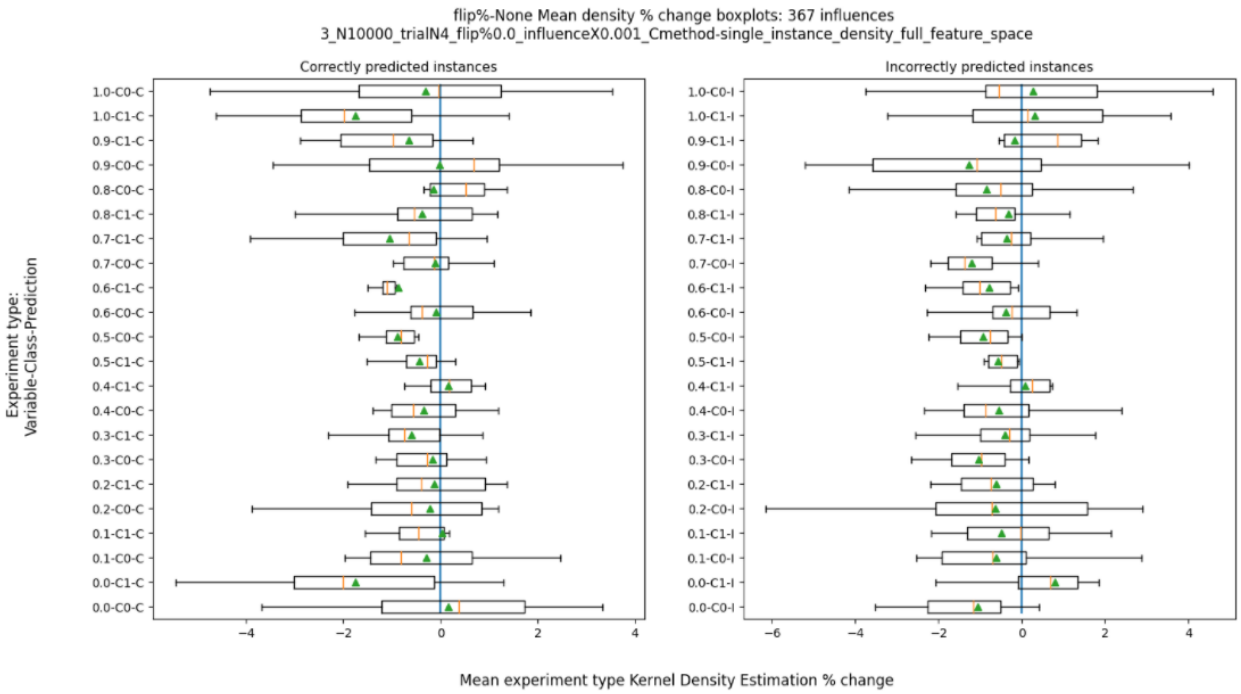


Figure 5.5: Flip% validation: on None influence drift

Thus, given the plot for this parameter set, it can be said that -5% and +5% are valid thresholds to classify the drift type as None instead of drift. When this drift strength threshold is applied, most of Type 1 false-positive errors will disappear. The None experiments that are partially correctly classified in Figure 5.5, would be classified as None instead.

However, it is better to signal that a drift is detected in the data in practice. The influence quantification also results in the detection. Thus, the advice can be given together with the drift conclusion that the quantification value is low. Then the domain expert of the machine learning algorithm can decide for himself if this drift strength is large enough for the problem or not.

Conclusion: A threshold should be determined for each parameter set and applied to account for the randomness of the experimental setup. This will give a better accuracy score with fewer false-positive rates in the None experiments.

Experimental setup parameter relationship

The setup of the experiment is done through multiple parameter settings. These parameters are the max_samples, n_trials, and influence_multiplication. When running many experiments, it was noticed that certain values of these parameters produce the same results. The following combinations are run: (1) Influence_multiplication grid search on fixed max_samples and fixed n_trials, (2) Influence_multiplication grid search with relative increase of max_samples and n_trials, and (3) n_trials grid search with fixed max_samples and fixed influence_multiplication to test the interdependent re-

Table 5.7: Variable parameter values for experimental relationship grid search

n_times	Max_samples	n_trials	Flip %	Method	Statistical test	Alpha
100	10.000	4	20%	Full_feature	Mann Whitney U	0.01

Table 5.8: Variable parameter values for inflX grid search on trial increase

n_times	Max_samples	n_trials	Flip %	Method	Statistical test	Alpha
10	10.000 - 100.000	4 - 40	0%	Full_feature	Mann Whitney U	0.01

lationship of these 3 parameters.

Grid search influence multiplication on fixed max_samples and n_trials.

Influence multiplication grid: [0.0001, 0.0005, 0.001, 0.005, 0.01] with fixed parameters in Table 5.7

Comparison: self-fulfilling vs self-defeating experiment.

Goal: Show the sensitivity of the detection approach to the inflX parameter.

Conclusion: There is no conclusive difference between the SF and the SD experiment. The higher influence power is detected in both experiments as the median relative KDE change is more prominent in the higher influence power experiments. However, the higher influence power also makes the outcomes more volatile and unreliable. This is because samples totally disappear from a feature space. This is the reason for the -100% reached density change in both experiments. This kind of drift is not realistic for a real-life algorithm, therefore, large influence multiplication values should not be used when testing an influence prediction drift detection and are thus further ignored in the method validation. The details of this mechanism are explained in the experimental validity section below.

Grid search influence multiplication on relative increase of n_trials and max_samples

Influence multiplication grid: [0.0001, 0.0005, 0.001, 0.005, 0.01] with fixed parameters in Table 5.8

Relative n_trials and max_samples change: x10

Goal: quantify the relationship of trials & samples on influence multiplication.

Table 5.9: Experimental setup relationship: 10k samples, 4 trials

influence_multiplication_grid	n_correct_experiments	n_partial_correct_experiments	n_incorrect_experiments	accuracy	false_negative_rate	false_positive_rate
0.0001	3	6	1	60.0%	30.1%	0%
0.0005	10	0	0	100.0%	0.0%	0%
0.0010	10	0	0	100.0%	0.0%	0%
0.0050	2	8	0	60.0%	40.0%	0%
0.0100	0	10	0	50.0%	50.0%	0%

Table 5.10: Experimental setup relationship: 100k samples, 40 trials

influence_multiplication_grid	n_correct_experiments	n_partial_correct_experiments	n_incorrect_experiments	accuracy	false_negative_rate	false_positive_rate
0.0001	10	0	0	100.0%	0.0%	0%
0.0005	10	0	0	100.0%	0.0%	0%
0.0010	10	0	0	100.0%	0.0%	0%
0.0050	5	5	0	75.0%	25.0%	0%
0.0100	0	10	0	50.0%	50.0%	0%

Conclusion: Increasing sample size and the number of trials 10x increases the low influence multiplication accuracy as seen in Figure **??**. This is seen in Table 5.9 and Table 5.10. This is most likely because the drift has more time to manifest itself through the increased samples, and intricate change details are better classified. The drift becomes unrealistic and biased in the low sample experiment with large influence multiplication values. This effect is not seen when 100k samples are used. Thus, the method is more robust when used with more samples and trials. This can be seen in Figure 5.6. Increasing inflX by 10x from 0.0005 to 0.005 roughly doubles both experiments' median KDE % change value. These specific values are only applicable with the current parameter set of Table 5.8.

Relationship conclusion: the parameters max_samples, n_trials, and influence_multiplications are interdependent variables that together determine how much time the drift has to manifest in the data. Localization: Choosing a trial length that is too big with an influenceX that is too low: effect of influence drift might take too long to manifest itself. So even when running experiments with drift, the density changes will be insignificant, and thus a false negative will be given. When choosing a small trial length with relatively large influenceX: the effect of influence will be too much. This is an unrealistic scenario. If an algorithm created data like this, it would be visually spotted that something is off. This detection method creates noise in the data calculation since trials with 0 samples at the start cannot get a density change calculation since their density was nonexistent at the start. Therefore, these instances are ignored, which skews the resulting metrics. This is explained through the visualization in Figure 5.8. When the regime flips at n=2500, there are no instances in both feature spaces left to start with a KDE estimation at T=0. Thus, there is nothing to compare to the change formula with the KDE at T=1. This issue can be challenged by making assumptions on the data, like always assuming a percentage of instances in that time position, so-called phantom instances. However, the choice was made not to apply this since a drastic drift like this is not realistic and would visually already create alarms of the algorithm user.

Experimental validity evaluation:

After doing all these implementation evaluations, the limitations and validities of the conceptual method should be discussed. This is the link between theory and observation. It is needed to discuss the validity threats to draw general conclusions. Conclusion validity: Certain design choices were made to improve the conclusion validity and create a more robust detection approach. Instead of using the less complex feature-split method, a more complex single_instance_kde method was chosen to provide more robust conclusions using a larger sample N. In addition, a non-parametric test was chosen not to assume a data distribution since the distribution is unknown a-priori. The treatment is chosen according to an apriori chosen unbiased checkerboard pattern.

The only conclusion validity issue that has to be addressed is the implementation of a high influence prediction drift. A strong drift causes instances to move to a certain feature space and disappear from another feature space. When these instances come back in the next trial, there were no instances to start within that trial. Thus, a KDE cannot be fit on the new instances. This makes the experiment biased towards density decreases. Internal validity: Causal relationship: Treatment causing the outcome?

Internal validity: Internal validity is the questioning of the causal relationship of the experiment. The main question here is to validate if the treatment is causing the outcome. This is the core problem this research tries to tackle. Current ML algorithms do not consider this validity because of simplicity reasons or the lack of tooling to account for it. The detection approach in this study is created to validate such experimental assumptions in an isolated environment. It still has to be assessed when this isolation is violated, and more causes for drift are introduced. However, some assumptions need to be valid for robust proof of the causal link between ML classification $\rightarrow$ Instance outcomes. One of these assumptions is the stable unit treatment value assumption (SUTVA). This assumes that the potential outcome of one instance should not be dependent on the treatment of another instance. If this assumption is violated, then the experiment has treatment interference. This research has treatment interference by design through one class instance being dependent on the other classes' classifications. This violation combined with prediction influence drift should be studied further.

Construct validity: The method's goal is to measure classification as a confounder to changing instances. While the method is validated on a relatively simple dataset, it is also designed with the possibility to extend the problem to multiple features and classes thereby tackling mono-operation

bias. Not only is it detecting that there is indirect prediction influence happening, but also giving a prediction about what kind of prediction influence. However, when this method is generalized to a real-world setting, confounding variables of the drifting data are not only the algorithm's prediction but can also be many other factors such as intrinsic drift and concept drift. These are not considered in this research, thus a gap in the current method. External validity: External validity is the most critical validity for this research. While this approach was validated in a controlled setting, it must still be proved useful in a generalized context. Unfortunately, time constraints did not allow us to validate this approach in more complex situations. This would test the more complex interactions that would portray between the dependent and independent variables. The next possible direct improvement is the validation of this method on a more complex dataset, so with more features, and more diverse feature values, and multiple classes. The step after that is to introduce intrinsic drift into the equation and validate the method on that. From there on, concept drift should be added to the experiment, and the detector should be compared to a concept drift detector. The last step would be to find real data where it is known that an algorithm was responsible for drift.

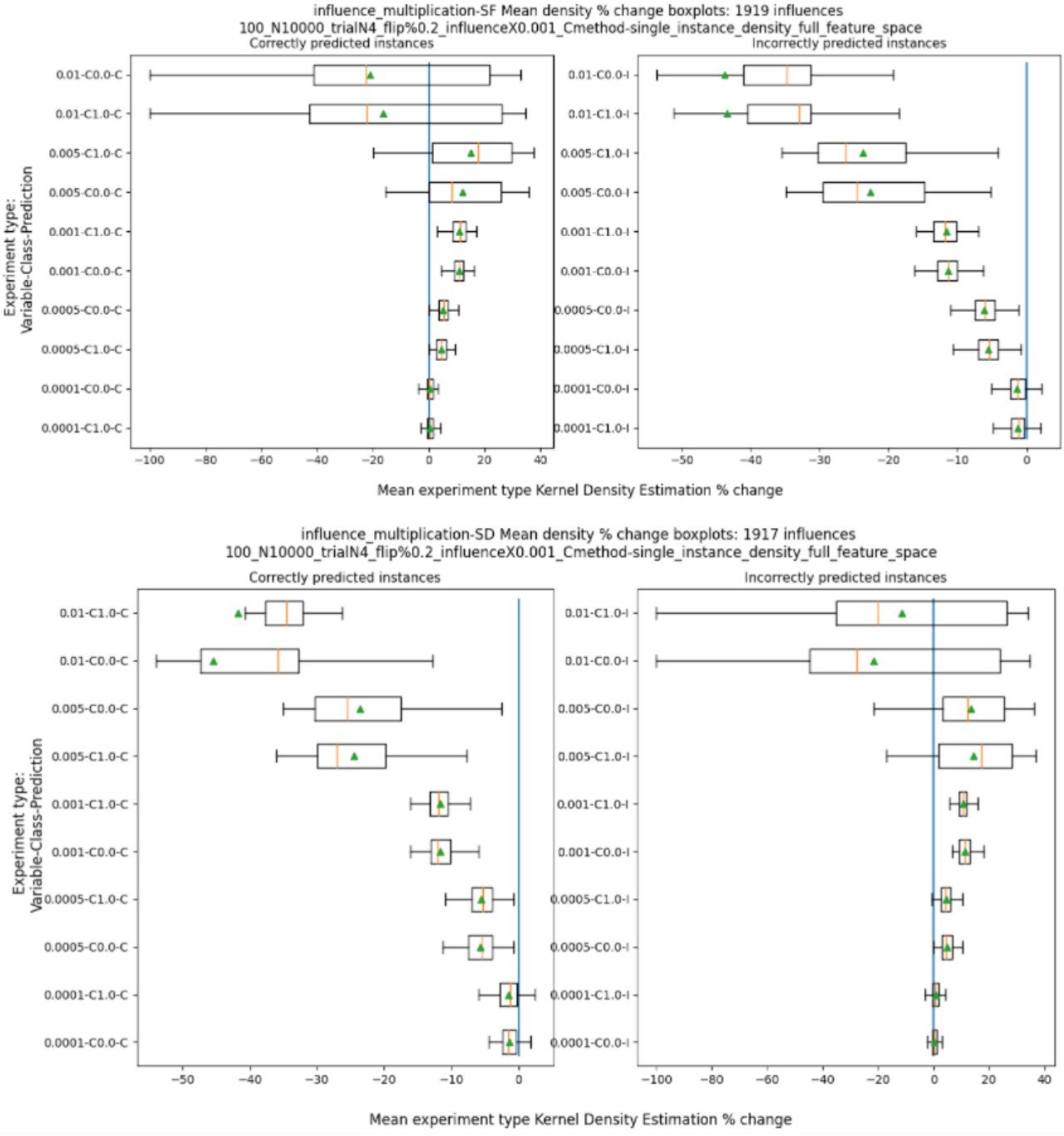


Figure 5.6: Experimental setup: relationship grid search: Self-fulfilling drift (top) vs self-defeating drift (bottom)

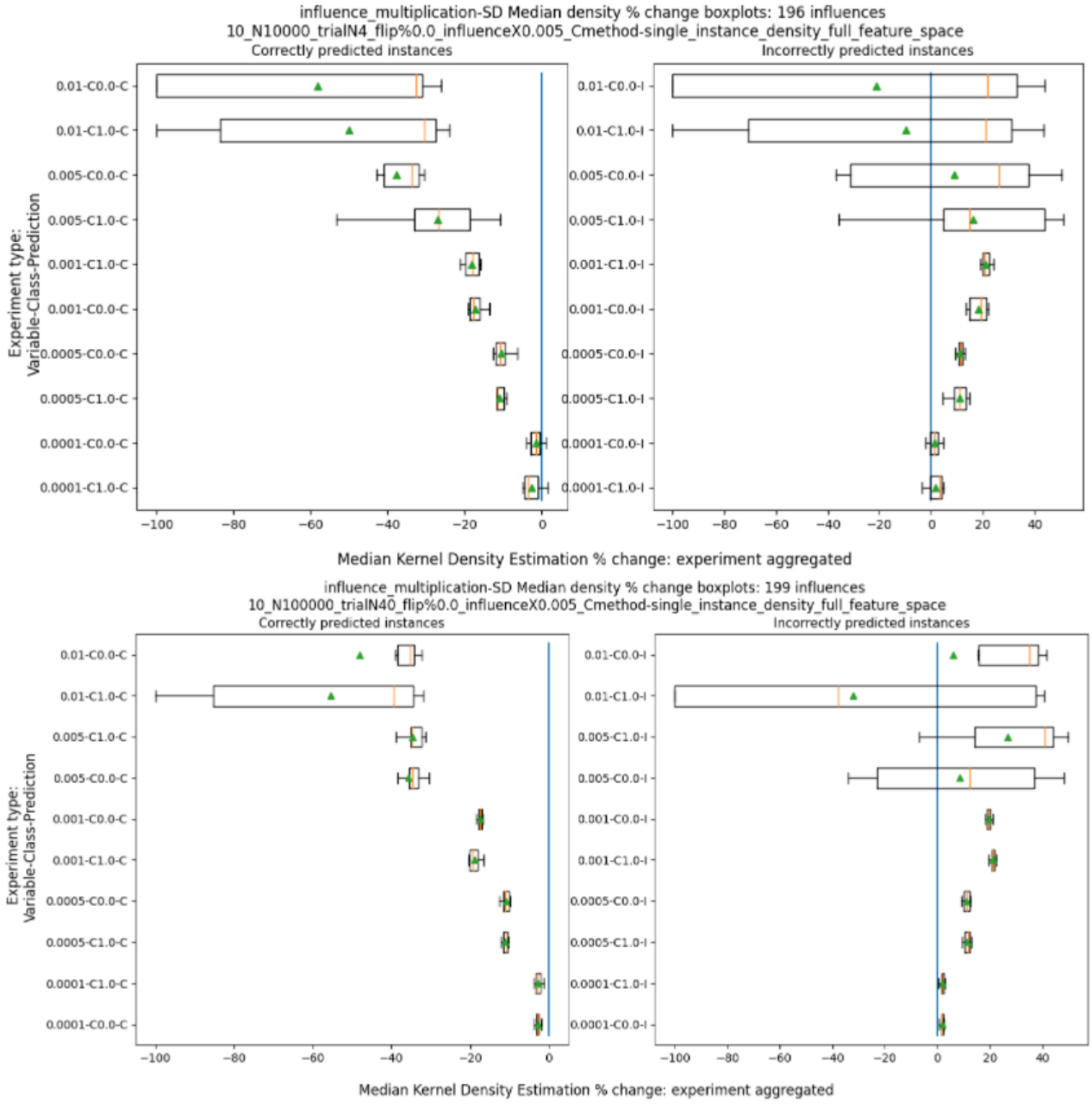


Figure 5.7: Experimental setup: relationship grid search: 10k samples, 4 trials (top) vs 100k samples, 40 trials (bottom)

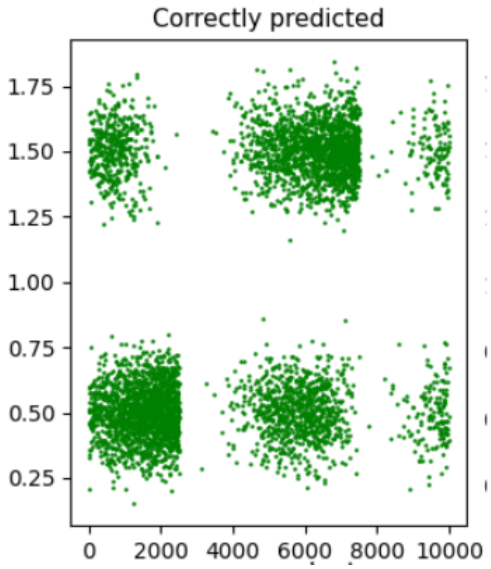


Figure 5.8: Positive classes over time with strong drift: instances dissapear

Chapter 6

Conclusion

Research question answers.

RQ1: What kind of feedback loops are created by systematic classifications systems?

A feedback loop categorisation is made through the use of a literature review. Different classification systems are mapped on how they influence the environment around them. A feedback loop can be categorised into several dimensions. These dimensions consist of (1) the influence mechanism, (2) the influence direction, (3) the influence consequences, (4) and the influence systems. The indirect feedback loop is chosen as the research target for the next research questions.

RQ2: What is the formal generation process behind the data chosen in RQ1 and how to generate such data?

Through the mappings of feedback loops done in research question 1, the inner workings of different feedback loops are better understood for this research. With this understanding, a dataset is created which generates data that can be influenced by an indirect feedback loop. A detection method can be created to classify and quantify this drift using this data.

RQ3: What kind of prediction drift detectors are currently available?

So far, there is only extensive research done on data drift detection and concept drift detection. There are no known methods found where a drift caused by a machine learning algorithm is detected, classified, and quantified.

This research has determined that prediction drift detection fundamentally is a causal problem. A shortlist of causal methods is reviewed. However, it was decided not implement the most popular methods, since to use them, domain knowledge of the problem is needed. These methods are also are complex to implement, and when linked with machine learning, this is also a novel topic with much

ongoing research.

Causal inference is linked with prediction drift through the use of structural causal models to understand a problem in a causal way. Knowing which variables influence other variables can be reasoned how drift would manifest in a system. Observational data is not enough to achieve this. Interventions need to be done to find out the causal structure of the problem. Therefore, the detection approach should make use of interventional techniques.

RQ4: How can prediction drift be detected in the feedback loop chosen in RQ1 by using causal methods?

In an indirect feedback loop, positive classifications increase the distribution of similar instances through network effects. In a self-defeating feedback loop, this effect is inverted. Causal methods can classify and quantify prediction influence drift through small interventions on the data and the calculation of the average treatment effect. These methods are tested in this research, and the outcome is a prediction drift detector that can detect drift on synthetic data.

The theoretical implications of this research are listed below:

In this research, feedback loops in domains were researched, categorisation of these feedback loops created, and these were linked to the domains found in research.

- An overview is created of causal inference methods related to feedback loops and the detection of influence drift.
- A practical understanding is created about how prediction influences drift detection.
- A method is designed to quantify influential prediction drift through causal inference techniques.
- Methods are created to evaluate prediction drift detection and the sensitivity of such detection.

Practical implications

- An open-source package is extended with a framework that future researchers can use to test different detection approaches on. This framework offers the possibility of creating streaming influence prediction drift datasets with variable self-fulfilling and self-defeating prophecies, applying any model to classify the streaming instances, and analysing the experiments' outcomes

through various visualisations and calculations. The hopes are that this framework will get extended further and the methods applied to real-world problems.

- The world is moving in the direction of regulating machine learning algorithms. This will result in the need to monitor and audit algorithms. The method created in this research is a start in that direction.

- A more holistic view of the algorithm can be provided by using different detectors together, like intrinsic drift, concept drift, and prediction influence detectors. This can result in an aggregated "Influence" metric, next to other drift metrics like "data drift metric" "concept drift metric". Alerts can further be based on those metrics, and possibly automatic actions can be triggered based on those alerts. For example, retraining the model or doing a sophisticated audit. These techniques are useful for a sophisticated MLOps monitoring and detection pipeline of potential influential algorithms.

- If prediction influence significance level can be shown with small randomisation of instance predictions, further testing with more robust parameter settings can be advised. This is advised to keep the costs low when starting. However, this is prone to Type 1 errors: predicting drift where there is no drift in reality. Initially, this is not an issue since the costs of making a Type 1 error are less than making a Type 2 error, predicting no drift when there is, in fact, a prediction influence drift.

Chapter 7

Future work

Due to the lack of research in the field of InfluenceML, and specifically prediction influence, future research needs to be done to crystalize these concepts and validate the results. Due to the time constraints of this thesis, several research directions were left unexplored. Follow-up research is encouraged further to build upon this framework through the following list of potential improvements is given here:

Different randomization techniques: The most important future research is to validate other randomization techniques. Implementing the technique used in the current approach did not deliver the wanted results. Suggestions to improve this could be to implement a different classification approach. First, a random classifier can be tried with a fraction of the instances with a known prediction beforehand or a learning model with a randomized fraction of instances.

A list of potential improvements for the current method:

Individual Treatment effect: Implement a detection extension method for individual instance influence quantification, which makes use of the individual treatment effect (ITE) instead of the current average treatment effect (ATE).

Detection and violation of SUTVA: This research violates the stable unit treatment value assumption through treatment interference. Therefore, the ATE calculation might be inaccurate. More research should be done into the causal assumptions of drift detection for more sophisticated drift quantification techniques.

Intrinsic and concept drift: Implement experiment setup extensions to take intrinsic drift and concept drift into account and validate the detection approach

Specific prophecies: Implement an experiment setup extension to apply more specific prophecies.

For example, only self-fulfilling drift in positive instances, localized drift in a certain feature area, or asymmetrical drift whereby correct classifications increase instances, but incorrect classifications do not decrease them.

Drift localization: Implement a detection extension to classify at which time point the drift is happening and at which time point it is not. An example could be: "In feature X of values around Z-Y, prophecy W is happening."

Temporal complexity: Implement an experiment setup extension to apply time to parameter settings. This can be a delay on arriving instances and a delay on the arrival of true class labels. However, this will be only relevant for real-time drift analysis instead of ex-post.

Experimental complexity: Validate the method using more complex experiment setups. This includes using more classes, more features, and diverse feature values.

When all that above deliver the wanted results, the step can be made to apply it on real-world data. However, this comes with all kinds of different challenges. The data need to get real-world data with instances + their according to feature values + model decisions. Alternatively, instances + their according to feature values + the model itself. The model can be applied and trained by researchers to produce the decisions. The issues of this are that their data is most likely the gold of their business for the company. This might get into trouble with the intellectual property rights of the complex model of the company. Another issue is the privacy of the people who are the data instances need to be respected.

Chapter 8

Acknowledgements

Foremost, I'm very thankful for the help and support that my supervisor, prof. habil. Georg Krempl gave me throughout the journey. Contrary to common thesis supervisor issues, he was always quickly available to discuss the progress, debate about fundamental topic concepts that I questioned, or offer tons of new directions and ideas to experiment with.

It was an honor to start this research with a company that saw potential in me and the outcomes of this research. However, when the relationship with the company deteriorated and difficult conversations needed to be had, Georg never showed any signs of doubt towards me or the research. His positivity, coaching, and belief motivated me in times where I did not believe in and could not motivate myself. From this experience I gained valuable wisdom about the inner workings of specific corporate norms and values, expectation management, and the industry-academic relationship.

I would also like to thank my second supervisor, prof. dr. A.P.J.M. Siebes, for his contributions and feedback on this work. His flexibility and quick reactions were invaluable for the progress of this research. Your willingness to read and examine this thesis is much appreciated.

My research colleague Tineke Jelsma also played an important role in this research. I want to thank her for her passion in the shared meetings we had together with Georg Krempl. She is the creator of some building blocks that my research used. She also freed up extra time to discuss the concepts and techniques her thesis was built upon.

My last thanks goes to all my friends and family who supported me. Special thanks go to Eline de Best, my girlfriend. She was my mental and emotional support throughout this process. Sometimes she helped me with concrete next steps and motivation, but mostly, she helped me by just being there for me.

Bibliography

Abdollahi, B. and Nasraoui, O. (2018). Transparency in Fair Machine Learning: the Case of Explainable Recommender Systems. In *Human and Machine Learning. Human–Computer Interaction Series.*, pages 21–35. Springer, Cham.

Adam, G. A., Chang, C.-H. K., Haibe-Kains, B., and Goldenberg, A. (2020). Hidden Risks of Machine Learning Applied to Healthcare: Unintended Feedback Loops Between Models and Future Data Causing Model Degradation. In *Proceedings of the 5th Machine Learning for Healthcare Conference*, volume 126, pages 710–731. PMLR.

Alpaydın, E. (2010). Introduction to Machine Learning Second Edition. Technical report, The MIT Press, Cambridge, MA.

An, W. (2018). Causal Inference with Networked Treatment Diffusion:. *https://doi-org.proxy.library.uu.nl/10.1177/0081175018785216*, 48(1):152–181.

Aneja, A. P., Avenancio-León, C. F., Dal Bó, E., Butler, A., Cook, P., Criscitello, D., Donohue, J. J., Goldin, J., Green, B., Grullón, G., Howard, T., Hoynes, H., Huck, J. R., Korgaonkar, S., Kulkarni, N., Kurakina, M., Levine, R., Lucas, D., Malmendier, U., Manso, G., Mccrary, J., Miller, C., Morse, A., Nguyen, H.-L., O'donnell, S., Pérez, M. C., Raphael, S., Ross, S. L., Sraer, D., Stanley, J., Stevenson, M., Walker, R., Wilcox, J., and Yuchtman, N. (2019). No Credit For Time Served? Incarceration and Credit-Driven Crime Cycles * Click for most recent version. Technical report.

Ang, H. H., Gopalkrishnan, V., Zliobaite, I., Pechenizkiy, M., and Hoi, S. C. (2013). Predictive handling of asynchronous concept drifts in distributed environments. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2343–2355.

Atwal, H. (2020). *Practical DataOps*. Apress.

Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. (2002). Models and issues in data stream systems. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–16.

Baena-García, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavaldà, R., and Morales-Bueno, R. (2006). Early Drift Detection Method. In *4th ECML PKDD International Workshop on Knowledge Discovery from Data Streams*, volume 6, pages 77–86.

Baier, L., Reimold, J., and Kühl, N. (2020). Handling Concept Drift for Predictions in Business Process Mining. *Proceedings - 2020 IEEE 22nd Conference on Business Informatics, CBI 2020*, 1:76–83.

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., and Herrera, F. (2020). Explainable Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115.

Bass, L., Weber, I., and Zhu, L. (2015). *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional.

Bemrose, B. (2019). Song length: the Spotify effect.

Bifet, A. and Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 7th SIAM International Conference on Data Mining*, pages 443–448. Society for Industrial and Applied Mathematics Publications.

Bottou, L., Peters, J., Quiñonero-Candela, J., Charles, D. X., Chickering, D. M., Portugaly, E., Ray, D., Simard, P., and Snelson, E. (2013). Counterfactual Reasoning and Learning Systems: The Example of Computational Advertising. Technical report.

Brzeziński, D. and Stefanowski, J. (2011). Accuracy updated ensemble for data streams with concept drift. In *Hybrid Artificial Intelligent Systems*, volume 6679, pages 155–163. Springer, Berlin, Heidelberg.

Cao, S. S., Jiang, W., Yang, B., and Zhang, A. (2020). How to Talk When a Machine is Listening: Corporate Disclosure in the Age of AI. *SSRN Electronic Journal*.

Capizzi, A., Distefano, S., and Mazzara, M. (2020). From DevOps to DevDataOps: Data Management in DevOps Processes. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12055 LNCS, pages 52–62. Springer.

Cerquitelli, T., Proto, S., Ventura, F., Apiletti, D., and Baralis, E. (2019). Towards a real-time unsupervised estimation of predictive model degradation. In *Proceedings of Real-Time Business Intelligence and Analytics*, New York, NY, USA. ACM.

Chaney, A. J. B., Stewart, B. M., and Engelhardt, B. E. (2018). How Algorithmic Confounding in Recommendation Systems Increases Homogeneity and Decreases Utility. In *Proceedings of the 12th ACM Conference on Recommender Systems*, New York, NY, USA. ACM.

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., and Wirth, R. (2000). CRISP-DM 1.0: Step-by-step data mining guide. Technical report, the CRISP-DM consortium.

Chen, Z. and Liu, B. (2018). Lifelong Machine Learning, Second Edition. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207.

Commission, T. E. (2020). On Artificial Intelligence-A European approach to excellence and trust White Paper on Artificial Intelligence A European approach to excellence and trust. Technical report, The European Commission.

Conover, M., Ratkiewicz, J., Francisco, M., Gonçalves, B., Menczer, F., and Flammini, A. (2011). Political Polarization on Twitter. In *Proceedings of the Fifth International Conference on Weblogs and Social Media*, Catalonia, Spain. aaai.

D'Alessandro, B., O'Neil, C., and Lagatta, T. (2017). Conscientious Classification: A Data Scientist's Guide to Discrimination-Aware Classification. *Big Data*, 5(2):120–134.

Dalvi, N., Domingos, P., Mausam, Sanghai, S., and Verma, D. (2004). Adversarial classification. In *KDD-2004 - Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 99–108. Association for Computing Machinery (ACM).

D'Amour, A., Heller, K., Moldovan, D., Adlam, B., Alipanahi, B., Beutel, A., Chen, C., Deaton, J., Eisenstein, J., Hoffman, M. D., Hormozdiari, F., Houlsby, N., Hou, S., Jerfel, G., Karthikesalingam,

A., Lucic, M., Ma, Y., McLean, C., Mincu, D., Mitani, A., Montanari, A., Nado, Z., Natarajan, V., Nielson, C., Osborne, T. F., Raman, R., Ramasamy, K., Sayres, R., Schrouff, J., Seneviratne, M., Sequeira, S., Suresh, H., Veitch, V., Vladymyrov, M., Wang, X., Webster, K., Yadlowsky, S., Yun, T., Zhai, X., and Sculley, D. (2020). Underspecification Presents Challenges for Credibility in Modern Machine Learning.

D'amour, A., Srinivasan, H., Atwood, J., Research, G., Baljekar, P., Sculley, D., and Halpern, Y. (2020). Fairness Is Not Static: Deeper Understanding of Long Term Fairness via Simulation Studies. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, New York, NY, USA. ACM.

Dandekar, P., Goel, A., and Lee, D. (2012). Biased Assimilation, Homophily and the Dynamics of Polarization. *Proceedings of the National Academy of Sciences of the United States of America*, 110(15):5791–5796.

Dang, Y., Lin, Q., and Huang, P. (2019). AIOps: Real-world challenges and research innovations. In *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion, ICSE-Companion 2019*, pages 4–5. Institute of Electrical and Electronics Engineers Inc.

Ensign, D., Friedler, S. A., Neville, S., Scheidegger, C., Venkatasubramanian, S., and Wilson, C. (2018). Runaway Feedback Loops in Predictive Policing. In *Proceedings of Machine Learning Research*, volume 81, pages 1–12.

Epstein, R. and Robertson, R. E. (2015). The search engine manipulation effect (SEME) and its possible impact on the outcomes of elections. *Proceedings of the National Academy of Sciences of the United States of America*, 112(33):E4512–E4521.

Fawcett, T. and Flach, P. A. (2005). A response to Webb and Ting's on the application of ROC analysis to predict classification performance under varying class distributions. *Machine Learning*, 58(1):33–38.

Ferraro, F., Pfeffer, J., and Sutton, R. I. (2004). Economics Language and Assumptions: How Theories Can Become Self-Fulfilling. *The Academy of Management Review 30*.

Frías-Blanco, I., Del Campo-Ávila, J., Ramos-Jiménez, G., Morales-Bueno, R., Ortiz-Díaz, A., and

Caballero-Mota, Y. (2015). Online and non-parametric drift detection methods based on Hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823.

Fursin, G., Guillou, H., and Essayan, N. (2020). CodeReef: an open platform for portable MLOps, reusable automation actions and reproducible benchmarking. Technical report, 1st Workshop on MLOps Systems.

Fuster, A., Goldsmith-Pinkham, P., Ramadorai, T., and Walther, A. (2017). Predictably Unequal? The Effects of Machine Learning on Credit Markets. *SSRN Electronic Journal*.

Galen, C. and Steele, R. (2020). Evaluating Performance Maintenance and Deterioration over Time of Machine Learning-based Malware Detection Models on the EMBER PE Dataset. In *2020 7th International Conference on Social Network Analysis, Management and Security, SNAMS 2020*. Institute of Electrical and Electronics Engineers Inc.

Gama, J., Medas, P., Castillo, G., and Rodrigues, P. (2004). Learning with drift detection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3171:286–295.

Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4).

Ginart, A., Zhang, E., Kwon, Y., and Zou, J. (2020). Competing AI: How does competition feedback affect machine learning?

Goldblum, M., Schwarzschild, A., Patel, A. B., and Goldstein, T. (2020). Adversarial Attacks on Machine Learning Systems for High-Frequency Trading.

Gomez-Uribe, C. A. and Hunt, N. (2015). The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems*, 6(4).

Hassanpour, N. and Greiner, R. (2019). Counterfactual regression with importance sampling weights. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2019-August, pages 5880–5887. International Joint Conferences on Artificial Intelligence.

Hinder, F. and Hammer, B. (2020). Counterfactual Explanations of Concept Drift.

Hu, H., Kantardzic, M., and Sethi, T. S. (2020). No Free Lunch Theorem for concept drift detection in streaming data classification: A review.

Hummer, W., Muthusamy, V., Rausch, T., Dube, P., El Maghraoui, K., Murthi, A., and Oum, P. (2019). ModelOps: Cloud-based lifecycle management for reliable and trusted AI. In *Proceedings - 2019 IEEE International Conference on Cloud Engineering, IC2E 2019*, pages 113–120. Institute of Electrical and Electronics Engineers Inc.

Jacobs, M., Pradier, M. F., McCoy, T. H., Perlis, R. H., Doshi-Velez, F., and Gajos, K. Z. (2021). How machine-learning recommendations influence clinician treatment selections: the example of the antidepressant selection. *Translational Psychiatry*, 11(1):108.

Khritankov, A. (2021). Analysis of hidden feedback loops in continuous machine learning systems. *Lecture Notes in Business Information Processing*, 404:54–65.

Kohavi, R., Longbotham, R., Sommerfield, D., Henne, R. M., Longbotham, R., Sommerfield, D., and Henne, R. M. (2009). Controlled experiments on the web: survey and practical guide. *Data Min Knowl Disc*, 18:140–181.

Krempl, G., Žliobaite, I., Brzeziński, D., Hüllermeier, E., Last, M., Lemaire, V., Noack, T., Shaker, A., Sievi, S., Spiliopoulou, M., and Stefanowski, J. (2014). Open challenges for data stream mining research. *ACM SIGKDD Explorations Newsletter*, 16(1):1–10.

Liu, L. T., Dean, S., Rolf, E., Simchowitz, M., and Hardt, M. (2019). Delayed impact of fair machine learning. *IJCAI International Joint Conference on Artificial Intelligence*, 2019-Augus(80):6196–6200.

Liu, L. T., Kalai, A. T., Wilson, A., Borgs, C., Haghtalab, N., and Chayes, J. (2020). The disparate equilibria of algorithmic decision making when individuals invest rationally. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 381–391. Association for Computing Machinery, Inc.

Madrid, J. G., Escalante, H. J., Morales, E. F., Tu, W.-W., Yu, Y., Sun-Hosoya, L., Guyon, I., and Sebag, M. (2019). Towards AutoML in the presence of Drift: first results. *AutoML*.

Malik, N. (2021). Does Machine Learning Amplify Pricing Errors in Housing Market? : Economics of ML Feedback Loops. *SSRN Electronic Journal*.

Mansoury, M., Abdollahpouri, H., Pechenizkiy, M., Mobasher, B., and Burke, R. (2020). Feedback Loop and Bias Amplification in Recommender Systems. *International Conference on Information and Knowledge Management, Proceedings*, 20:2145–2148.

Merton, R. K. (1948). The Self-Fulfilling Prophecy. *The Antioch Review*, 8(2):193.

Miller, J., Milli, S., and Hardt, M. (2020). Strategic Classification is Causal Modeling in Disguise. In *Proceedings of the 37th International Conference on Machine Learning*, pages 6917–6926. PMLR.

Mohanty, S. and Vyas, S. (2018). IT Operations and AI. In *How to Compete in the Age of Artificial Intelligence*, pages 173–187. Apress.

Montiel, J., Halford, M., Mastelini, S. M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A., Gomes, H. M., Read, J., Abdessalem, T., and Bifet, A. (2021). River: Machine learning for streaming data in python. *Journal of Machine Learning Research*, 22(110):1–8.

Mooij, J. M. and Claassen, T. (2020). Constraint-based causal discovery using partial ancestral graphs in the presence of cycles. *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence, UAI 2020*, 36(124):1159–1168.

Mooij, J. M., Peters, J., Janzing, D., Zscheischler, J., Schölkopf, B., Guyon, I., Statnikov, A., Mooij, M., and Mooij, S. (2016). Distinguishing Cause from Effect Using Observational Data: Methods and Benchmarks. *Journal of Machine Learning Research*, 17:1–102.

Moreno-Torres, J. G., Raeder, T., Alaiz-Rodríguez, R., Chawla, N. V., and Herrera, F. (2012). A unifying view on dataset shift in classification. *Pattern Recognition*, 45(1):521–530.

Muchnik, L., Aral, S., and Taylor, S. J. (2013). Social Influence Bias: A Randomized Experiment. *Science*, 341(6146):647–651.

Nowak, A., Ross, A., and Yencha, C. (2018). SMALL BUSINESS BORROWING AND PEER-TO-PEER LENDING: EVIDENCE FROM LENDING CLUB. *Contemporary Economic Policy*, 36(2):318–336.

Oliveira, G., Minku, L., and Oliveira, A. (2021). Tackling Virtual and Real Concept Drifts: An Adaptive Gaussian Mixture Model. Technical report.

Oza, N. C. and Russell, S. (2001). Experimental comparisons of online and batch versions of bagging and boosting. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 359–364. Association for Computing Machinery (ACM).

Pan, J., Pham, V., Dorairaj, M., Chen, H., and Lee, J.-Y. (2020). Adversarial Validation Approach to Concept Drift Problem in User Targeting Automation Systems at Uber. Technical report.

Pearl, J. (2009). *Causality: Models, reasoning, and inference, second edition*. Cambridge University Press.

Pearl, J. (2018). Theoretical Impediments to Machine Learning With Seven Sparks from the Causal Revolution. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 3–3, New York, NY, USA. ACM.

Pearl, J. and Verma, T. S. (1995). A theory of inferred causation. *Studies in Logic and the Foundations of Mathematics*, 134(C):789–811.

Quiñonero-Candela, J., Sugiyama, Masashi Schwaighofer, A., and D. Lawrence, N. (2009). *Dataset Shift in Machine Learning*, volume 173. The MIT Press, Cambridge, MA.

Read, J. (2018). Concept-drifting Data Streams are Time Series; The Case for Continuous Adaptation. Technical report.

Rosenbaum, P. R. and Rubin, D. B. (1983). The Central Role of the Propensity Score in Observational Studies for Causal Effects. *Biometrika*, 70(1):41.

Rubin, D. B. (1973). Matching to Remove Bias in Observational Studies. *Biometrics*, 29(1):159.

Schlimmer, J. C. and Granger, R. H. (1986). Beyond incremental processing. *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, pages 502–507.

Scholkopf, B., Locatello, F., Bauer, S., Ke, N. R., Kalchbrenner, N., Goyal, A., and Bengio, Y. (2021). Toward Causal Representation Learning. *Proceedings of the IEEE*, 109(5):612–634.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Philips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. (2015). Hidden technical debt in Machine learning systems. In *NIPS'15: Proceedings of the 28th International Conference on Neural Information Processing Systems*, pages 2503–2511.

Sethi, T. S. and Kantardzic, M. (2018). Handling Adversarial Concept Drift in Streaming Data. *Expert Systems with Applications*, 97:18–40.

Shanbhag, A., Ghosh, A., and Rubin, J. (2021). Unified Shapley Framework to Explain Prediction Drift.

Shaughnessy, J. and Zechmeister, Eugene Zechmeister, J. (2015). *Research Methods in Psychology*. McGraw-Hill Education, New York: McGraw-Hill.

Sinha, A., Gleich, D. F., and Ramani, K. (2016). Deconvolving Feedback Loops in Recommender Systems. *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 3251–3259.

Smith, A., Toor, S., and Kessel, V. P. (2018). Many Turn to YouTube for Children's Content, News, How-To Lessons | Pew Research Center.

Söllner, M., Hoffmann, A., and Leimeister, J. M. (2016). Why different trust relationships matter for information systems users. *European Journal of Information Systems*, 25(3):274–287.

The New Economy (2017). The troubling influence algorithms have on how we make decisions – The New Economy.

Voortman, M., Dash, D., and Druzdzel, M. J. (2010). Learning why things change | Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence. In *UAI'10: Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pages 641–650.

Wager, S., Chamandy, N., Muralidharan, O., and Najmi, A. (2013). Feedback Detection for Live Predictors. *Advances in Neural Information Processing Systems*, 4(January):3428–3436.

Wang, H. and Abraham, Z. (2015). Concept drift detection for streaming data. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2015-September. Institute of Electrical and Electronics Engineers Inc.

Weimer, J. (2019). Dataset Shifts in Autonomous Systems CIS700: Safe Autonomy.

Widmer, G. and Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101.

Wieringa, R. J. (2014). *Design science methodology: For information systems and software engineering*. Springer Berlin Heidelberg.

Yu, H., Liu, T., Lu, J., and Zhang, G. (2021). Automatic Learning to Detect Concept Drift.

Zhang, Y., Feng, F., He, X., Wei, T., Song, C., Ling, G., and Zhang, Y. (2021). Causal Intervention for Leveraging Popularity Bias in Recommendation. 10.

Žliobaitė, I. (2010). Learning under Concept Drift: an Overview. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346 – 2363.

Žliobaitė, I., Pechenizkiy, M., and Gama, J. (2016). An Overview of Concept Drift Applications. In *Big Data Analysis: New Algorithms for a New Society*, pages 91–114. Springer, Cham.