Evaluating Data Distribution Based Concept Drift Detectors

Konsta Kanniainen[1]

Supervisors: Jan Rellermeyer[1]**, Lorena Poenaru-Olaru**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 29, 2023

Name of the student: Konsta Kanniainen
Final project course: CSE3000 Research Project
Thesis committee: Jan Rellermeyer, Lorena Poenaru-Olaru, Jesse Krijthe

Abstract

Various techniques have been studied to handle unexpected changes in data streams, a phenomenon called concept drift. When the incoming data is not labeled and the labels are also not obtainable with a reasonable effort, detecting these drifts becomes less trivial. This study evaluates how well two data distribution based label-independent drift detection methods, SyncStream and Statistical Change Detection for Multi-Dimensional Data, detect concept drift. This is done by implementing the algorithms and evaluating them side by side on both synthetic and real-world datasets. The metrics used for synthetic datasets are False Positive Rate and Latency; for real-world datasets, Accuracy is used instead of Latency. The experiments show that both drift detectors perform significantly worse on real-world than on synthetic data.

1 Introduction

Machine learning models that are deployed to handle a stream of new incoming data after an initial training often suffer from a drop in performance over time [1]. One reason for this is that the data used in the initial training of the model is no longer representative of the current state of the data source. This is called concept drift. One way to check for concept drift is to simply look for drops in model accuracy, but this assumes labels to be available at the time of evaluation, which may be costly, inconvenient, or even impossible. Label-independent concept drift detectors aim to tackle this problem by various techniques, but are so far not researched as extensively as label-dependent detectors [2].

Data distribution based drift detectors consider the distribution of the input data and detect changes in it [2]. This is done by comparing recent data to reference data with a distribution distance measure [2], [3]. Depending on the technique, a method to estimate the expected spread of the data may be needed to compute a critical region for the test statistic [3], [4]. Song et al. have proposed a distance measure as well as a variance estimation procedure for multi-dimensional data [4], and parts of the SyncStream algorithm also use distribution difference metrics to detect drifts [5]. What these studies have in common is the lack of published implementations for reproduction and further research.

The research question evaluated in this study is:

- How well do data distribution based concept drift detectors identify concept drift in case of synthetic/real-world data?

An important step in approaching the research question is to select data distribution based concept drift detectors for which no implementations are currently available and implement them. Enabled by this, the main contributions of this paper are the performance measurements and side-by-side comparisons of data distribution based drift detectors on both synthetic and real-world data. The implementations are also published[1].

[1]https://github.com/konstaka/cse3000-research-project

The related works, including drift detectors implemented in this study, are explored in Section 2. Section 3 walks through the implementations of the selected detectors and describes the setup, and Section 4 contains results from the performance experiments. Section 5 discusses the results and weighs them against existing research. Section 6 concludes the paper with a recap of the findings and contributions.

2 Related Work

Concept drift detectors can roughly be divided into two categories; label-dependent and label-independent [1]. Data distribution based detectors belong to the label-independent category, the distinguishing feature being that they do not need ground true labels to function. Label-independent drift detectors are typically more expensive computation-wise [2]. One common reason to research and employ them anyway is that, in practice, true labels are in many cases expensive or impossible to collect in a reasonable time frame [1].

A recent survey on concept drift detectors examined over 130 different algorithms [2]. In addition to fully label-dependent and fully label-independent algorithms, the survey identified a category of semi-supervised drift detectors, in which some incoming data is provided a ground true label to help evaluating the drift. It was concluded that label-independent and semi-supervised drift detectors have been studied relatively little compared to label-dependent ones.

In the process of determining drift by measuring data distribution change, two sets of data are needed for comparison. This typically means defining a fixed reference window and a sliding test window over the data [2]. The exact setup of these windows, however, is either algorithm-specific or left to the user to decide on; they might overlap, or both of them might slide. After determining the windows to compare, a dissimilarity test is then applied to them and the significance of the result is assessed to determine if it meets the desired criteria. Both of these steps can be done in several ways.

Several data distribution based concept drift detectors have been published [2], of which we selected SyncStream and Statistical Change Detection for Multi-Dimensional Data (SCD) to be implemented in this study.

2.1 SyncStream

SyncStream is a data stream mining algorithm that stores past prototypes in a special tree structure [5]. In addition to this, it has two strategies for explicitly detecting abrupt concept drift. The first strategy employs principal component analysis to reduce the covariance matrix of each window of data into a principal eigenvector. The angle between the eigenvectors of two consecutive windows is calculated and a drift is reported if this angle is sufficiently large. 60 degrees is used by the authors as a threshold. The second strategy is a generalized Wilcoxon test, also called the Brunner Munzel test. It ranks the data by each feature in both windows separately and in the unified window and examines the differences in these ranks. With window sizes of over 20 data points, this test statistic follows the standard normal distribution, and a user-supplied p-value can directly be used to define significance. The authors use p=0.01 (two-tailed). These strategies are combined with an OR-clause to determine drift.

2.2 SCD

SCD, also referred to as the density test, takes aside a randomly selected half of the fixed reference window and trains a kernel density estimator on it, defining a kernel at each data point but with the restriction that no data point is considered to be generated by its own kernel [4]. In order to get a good model, the bandwidths are optimised individually by a special expectation maximization process defined in the paper. The resulting estimator is then tried on the remaining half of the reference window and the test window, respectively. Put simply, in this algorithm concept drift is defined as the difference between the likelihoods of these windows of data being generated by the trained estimator. The significance of this difference is determined by bootstrapping the remaining half of the reference window to obtain sample variances, which are used to estimate the variance of the underlying data distribution. With this, the critical value for the test statistic is determined based on a given p-value; p=0.08 is used by the authors. The test is designed to be run in both directions for maximum power, in which case the p-value is halved for both runs, but the reverse direction is only tried for a batch if the test in the main direction does not already find a drift in it.

3 Methodology

3.1 Implementations of concept drift detectors

SyncStream adaptations

The drift detection methods of SyncStream are included as part of the full algorithm, considering two consecutive test batches at a time [5]. The first test batch is compared to the reference data. SyncStream applies both of the techniques on label-wise collections and tests for distribution changes within each label. Since implementing the full SyncStream algorithm was not in scope for this study, and we assume that labels are not available in the test data, we needed a way around this requirement.

The first attempt at tackling this included training a simple classifier that would predict a label for each item in a test batch before the drift detectors examined it. This was tried only on the synthetic data, each type of dataset requiring a different classifier. One dataset was also experimented with in a supervised manner, relaxing the assumption of unavailable labels. While the results obtained with these experiments are included for completeness, another, improved approach came up later, keeping the process label-independent.

The second approach was to leverage the idea that as the SyncStream methods were designed to detect distribution changes within the labels, it could be viable within the original, combined stream as well. This lead to applying the drift detection techniques on whole consecutive test batches without grouping them by label first.

Finally, the framework for evaluating the detectors on real-world datasets required that the batches are measured one by one against a fixed reference batch. Due to this, the final adaptation of both SyncStream methods was to compare all test batches against the reference data instead of the previous test batch. A parameter was included in the function to switch between these two modes, and both of them are evaluated on the synthetic datasets.

SyncStream: PCA

We used PCA from the scikit-learn Python package[2] to get the principal eigenvector of the feature covariance matrix of each test batch and computed the angle change between consecutive batches using NumPy [6]. In the original study, 60 degrees was used as a threshold value for detecting a drift [5], but after experiments on our synthetic datasets, we chose 30 degrees as a threshold, effectively calibrating this method on the SEA dataset. This angle represents how much the covariances of the feature pairs shift between the batches. The difference in the observed suitable threshold may be due to the datasets used or simply a byproduct of adapting the method outside of its intended context as part of an online learning algorithm.

SyncStream: Wilcoxon test

The adapted version of the Wilcoxon test used in SyncStream was not found to be present in available libraries, so most of it had to be implemented from scratch. This included methods for computing a midrank, the difference in midranks between the batches, and the feature-wise rank difference sum within a batch. Rankdata from the SciPy package[3] was used to compute feature-wise ranks for the data. To define significance, the same two-tailed p-value of 0.01 was used as in the original study [5]. As the test statistic follows a standard normal distribution, this effectively meant setting a threshold value of 2.575 for the absolute value of the test statistic.

SCD

The size of most datasets in the original SCD study were in the scale of thousands rather than tens or hundreds of thousands like in our study [4]. This meant that the expectation maximization process for estimating individual bandwidths for each kernel proved to be too computationally expensive for the scope of this paper. Instead, we chose the Scott method, which the authors compared their process to, to be used as a drop-in replacement. This enabled the use of an existing library implementation of kernel density estimation, namely gaussian_kde from the SciPy package[4]. Other SciPy and scikit-learn functions were also used where applicable throughout the implementation. A preliminary implementation of the expectation maximization procedure, while abandoned from the evaluations due to the performance issues, is included in the published code for possible future use.

Further, in the interest of making the algorithm run faster still, the estSize parameter was toned down from the original 4000 to 30, with the additional restriction that the standard deviation of the variance estimates changes less than 1% between iterations [4]. This may reduce the quality of the variance estimate, namely by making the bootstrap percentile method work with less estimates to pick from. However, this also makes the expensive variance estimation step, which has to be done once per reference batch, run significantly

[2]scikit-learn version 1.0.2 https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[3]scipy version 1.9.1 https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.rankdata.html

[4]scipy version 1.9.1 https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.gaussian_kde.html

faster. Since the detector produced a perfect score on the SEA datasets even with this change and the distribution of the estimates always settled within a few iterations after 30 estimates in our experiments, we consider the effect negligible for our purposes. The expensive part of the step is parallelizable, so this is something worth revisiting in possible future implementations.

SCD faced limitations with some datasets. With Airlines, it could not run at all on a laptop with a 6-core Intel i7-9750H processor and 16 GB of memory; the Python3 kernel consistently died while trying. We note that this detector, due to the expensive kernel density estimator it uses, is not well suited for very large datasets.

With the other real-world datasets, linear algebra errors emerged. To address these, a PCA preprocessing step was used on Elect2 and on the bidirectional runs of Weather without reducing dimensions. The same problem arose on the AGRAW datasets with one-hot encoding, and the last experiments showed that this method should also work there. Due to the sparse nature of the Spam dataset, dimensions had to be reduced even further than the rank of the training data matrix (1276) for the detector to work even in one direction. We used PCA from sklearn.decomposition, with n_components=100 for Spam. Furthermore, the experiments on Spam are made with an inverted test statistic, as this arose as an interesting pattern during the experiments.

For some experiments, SCD was only run in one direction due to computation costs. Bidirectional mode uses more time to train a KDE on each test batch and thus it was used when the initial experiments showed that the result could be improved by enhancing the power of the test, which is what the bidirectional mode of SCD was designed to do [4].

3.2 Description of datasets

See Table 1 for the dataset dimensions.

Dataset	# rows	# features
SEA	100000	3
AGRAW1	100000	9
AGRAW2	100000	9
Airlines	539383	4
Elect2	45312	8
Weather	18159	8
Spam	4405	10727

Table 1: Dataset dimensions.

Synthetic datasets

Three synthetic datasets, **SEA**, **AGRAW1**, and **AGRAW2**, as described in [1], were used for developing and evaluating the drift detectors. Each contains 100,000 rows, generated so that each dataset contains a concept drift starting at row 55,000. In addition to abrupt concept drift, versions with gradual drifts of sizes 500, 1000, 5000, 10000, and 20000 rows were used.

Of these datasets, SEA only contains numerical features, while both the AGRAW datasets contain numerical as well as categorical features. The categorical features were encoded with TargetEncoder from the category_encoders Python package[5], as well as OneHotEncoder and OrdinalEncoder from scikit-learn[6], using the training data for fitting. Further, all features were scaled to between 0 and 1 using MinMaxScaler from scikit-learn[7]. Some experiments without the scaler are included for comparison.

[5]category_encoders version 2.5.1.post0 https://contrib.scikit-learn.org/category_encoders/

[6]scikit-learn version 1.0.2 https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing

[7]scikit-learn version 1.0.2 https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

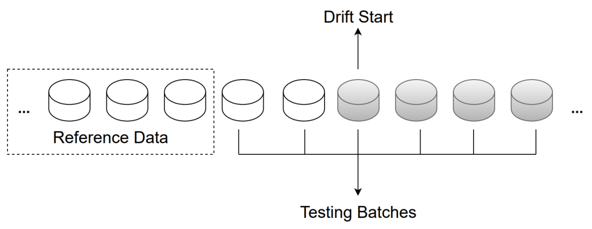

Figure 1: Experiment setup. [1]

Real-world datasets

Four real-world datasets were used; **Airlines** [7], **Elect2** [8], **Weather** [9], and **Spam** [10]. Airlines contains both numerical and categorical features, while Elect2, Spam, and Weather only contain numerical features. Airlines is a dataset of flight details, labeled by if the flight was delayed or not. Elect2 contains electricity market data, predicting if the price is going up or down on a given day. Weather contains daily weather details and is labeled by whether it was raining or not. Spam is a collection created from the spam assassin corpus, each column corresponding to a word and each row representing an e-mail, labeled by if it was spam or not. Categorical features were encoded using all three encoders, and all datasets were tried with and without MinMaxScaler.

3.3 Data setup

To set the stage for the experiments, all datasets were split into reference data and testing batches to simulate consuming a data stream after initial training. See Fig. 1 for illustration of the data setup. Each synthetic dataset was known to start drifting at row 55,000; for real-world datasets, a process examining model accuracies was used to determine the drifts.

The first 30% of each synthetic dataset was taken as reference data. Within this subset we could assume that the labels were known and use them for the purposes of training and defining baselines. The rest of the data was divided into equal-sized test batches with each containing 10% of the whole dataset. Each of the synthetic datasets was therefore known to start drifting in test batch 3.

For the real-world datasets, batching was dataset-specific and defining the drifts was done by scoring the test batches on classifiers trained on the respective training sets. Threshold for deciding if the batch contained a drift was determined by either cross-validation or time-based splitting; one standard deviation was removed from the expected accuracy.

For Airlines, the first 179794 rows were used for training, and the batches were to be 17000 rows long to represent daily batches. The classifier used was KNeighborsClassifier from sklearn.neighbors. Time-based splitting was used to determine expected accuracy, this is considered a preferred method over cross-validation [11]. A modified cross-validation technique using sequential rows for testing was initially used, but abandoned due to inconsistent and irreproducible results. Thus, four batches were removed from the end of the training data to be used for validation and the features were scaled. With all three encoders, only batch 1 was classified not drifting for the purposes of this study. This could be an indication of model overfitting, so we verified this on RandomForestClassifier from sklearn.ensemble. These experiments showed notable overfitting; the model accuracy was much higher on the part used for training than on the rest of the reference data. This pattern was absent with the kNN model, so we assume that the accuracy drop for nearly all test batches of Airlines is more likely due to concept drift than overfitting. See Fig. 2 for illustration.

For Elect2, the first 15104 were used for training. Since each row represents a day, a batch size of 365 meant yearly batches. After comparing multiple classifiers, the one selected here was AdaBoostClassifier from sklearn.ensemble. A modified cross-validation technique with 365 random rows left for validation each time was used, and the defining result was an average of 40 runs. Features were scaled. This gave a mix of drifting and non-drifting batches throughout, with a notable long stretch of drifting batches in the middle of the data, suggesting a recurring concept drift.

With the Weather dataset, we employed two batching strategies; monthly and yearly, corresponding to batch sizes of 30 and 365. The dataset was examined with a multitude of classifiers, SVC from sklearn.svm being selected for the definitions as it was the best performer on the yearly batches. The above described cross-validation method was similarly used here, taking an average of at least 10 runs and until the standard error of the measured accuracies dropped under 0.05. The first 6053 rows were used for training and the features were scaled. A mix of drifting and non-drifting batches was, again, found throughout the testing data for both monthly and yearly batches, like in the Elect2 dataset.

For the Spam dataset, RandomForestClassifier was used after trying out a few others. The features were not scaled. The first 1468 rows were used for training and batch sizes were set to 100, 50, and 20 for three different experiments. In each batching strategy, the modified cross-validation strategy yielded notably different results than time-based splitting, the latter classifying significantly more batches as drifts. While both are included in the published implementations, we chose the versions with time-based splitting for these definitions, as it is expected to be produce more accurate results [11]. The drift behavior of Spam resembles that of Airlines, in that the model initially performs decently but exhibits a consistent trend of decreasing accuracy over time.

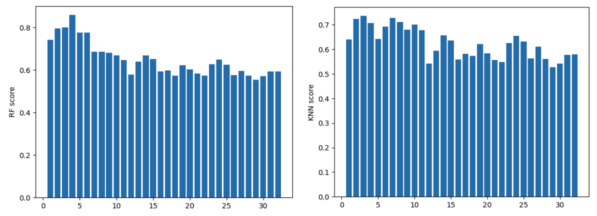

Figure 2: Experiments were done to rule out overfitting in the kNN model used to determine drifts in Airlines. The first 6 batches were used for training. The random forest model on the left shows notable overfitting, whereas the kNN model on the right does not.

3.4 Evaluation metrics

Synthetic data

To evaluate performance in terms of false alarms, we use **False Positive Rate** (FPR_s). This is the number of test batches erroneously flagged as containing a drift before the drift actually happened, divided by the total amount of non-drifting test batches before the (first) drift in the dataset [1]. In other words, FPR_s is the ratio at which the detector flags non-drifting batches before any drift occurs.

The other measure for synthetic datasets in this study is **Latency**. This is defined as the distance in batches between the first test batch flagged as containing a drift at of after the drift occurs and the batch actually containing the drift, divided by the amount of batches remaining in the dataset after the start of the drift [1]. If no drift is ever detected or is only detected before the drift actually happened (false positive), latency is considered ”Not Detected” and encoded as 1 for computations. Detecting only the last test batch in a dataset also results in a latency of 1; we consider these results equal in terms of usefulness of the detector.

Ideally, FPR_s and latency would both be as low as possible. An example of a dataset of 7 test batches containing an abrupt concept drift in batch 3 and the detector flagging batches 2, 3, and 4 would have a false positive rate of 0.5 (batch 1 was not flagged, but batch 2 was, despite not containing a drift) and a latency of 0 (drift was detected on time at batch 3). Another example of a similar dataset with 7 batches but containing a gradual drift that spans over batches 3 and 4, with the detector flagging only batch 4, would result in a FPR_s of 0 and a latency of 0.25 (as the detection was late by one batch from the start of the drift with 4 batches left after it in the dataset). This study is limited to producing these measurements on the selected drift detectors and datasets. The appropriate thresholds for them are therefore out of scope.

Real-world data

In real-world datasets we recognise that there may be more than one drift in the data. Every test batch is therefore evaluated with the detectors and flagged as drift or no drift. **False Positive Rate** (FPR_{rw}) is again used, but it now means the ratio of erroneously flagged drifts to the total number of known non-drifting batches as defined by the accuracy computation methods. The other measure for real-world datasets, **Accuracy**, is defined as the ratio of correctly detected drifts to all known drifts. So while FPR_{rw} is still ideally low, accuracy is now desired to be as high as possible.

Method	SEA	Agraw1	Agraw2
PCA (kNN)	(0.5, 0.0)		
PCA (kNN, 5000)	(0.6, 0.0)		
PCA (kNN, gradual)	(0.5, 0.25)		
PCA (RF)		(0.0, 1.0)	(0.0, 1.0)
Wilc. (RF)		(0.0, 1.0)	(0.0, 1.0)
PCA (supervised)	(0.5, 0.0)	(0.0, 1.0)	(0.0, 1.0)
PCA (sv., gradual)	(0.5, 0.25)		

Table 2: Drift detection performance on the synthetic datasets, measured in False Positive Rate and Latency (FPR_s, L). These experimental methods are included here for completeness. They all use the previous test batch as reference, and are evaluated on abrupt drifts, except two on gradual drifts (20000 rows). The features were not scaled, and the encoder was always TargetEncoder. The models used to simulate the SyncStream environment were KNeighborsClassifier with k=13 and RandomForestClassifier with default parameters. One experiment was done on batch size of 5000 instead of the default 10000 to rule out the effect of partly drifting batches.

4 Results

Evaluation results are shown in the tables. For SyncStream methods, *PCA* and *Wilc.* are used as labels for the PCA strategy and the rank sum test strategy respectively. Results with the fixed reference adaptation are marked with *fixed* and the original, continuous version with *cont.*. Different encoders are marked with *ohe*, *oe*, and *te* for OneHotEncoder, OrdinalEncoder and TargetEncoder respectively and experiments without MinMaxScaler are marked with *u*. The metrics are combined in the same cells as (FPR_s, L) for synthetic and (FPR_{rw}, Acc) for real-world datasets. Cells are left empty in case of missing results, or, in case of the SEA dataset, not using encoders. The final results were converted from the initial Python output into LaTeX tables using Node.js scripts.

Experiments on preliminary SyncStream implementations that were tried with the synthetic datasets but later abandoned in favor of better functioning ones are displayed in Table 2. They include two methods of predicting labels for grouping the test batches by label. One of the methods was also tried with a different batch size. Finally, the PCA method was also tried in a supervised manner, so the test labels were used for grouping. These methods resulted in arbitrary behavior of the test statistics and were thus unusable.

4.1 Synthetic data

Abrupt drift

Table 3 displays the detectors' drift detection performance on abruptly drifting synthetic datasets. This was the starting point of the experiments, and for completeness it shows more of the methods that were initially tried before moving on to other datasets. Unscaled versions were mostly not included further, as we found almost no difference in the initial runs and using the scaler would be expected to work better [1]. Initial results also lead us to abandon one-hot encoding for the Wilcoxon test and ordinal encoding for the others.

The drift in the SEA dataset could be detected by all detectors, while the AGRAW datasets proved to be more difficult. The PCA method does not work on them at all. The Wilcoxon

Method	SEA	Agraw1	Agraw2
PCA (cont., oe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (cont., oe)		(0.0, 1.0)	(0.0, 1.0)
PCA (cont., ohe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (cont., ohe)		(1.0, 0.0)	(1.0, 0.25)
PCA (cont., te, u)	(0.0, 0.25)	(0.0, 1.0)	(0.0, 1.0)
PCA (cont., te)	(0.0, 0.25)	(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, oe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, oe)		(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, ohe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, ohe)		(1.0, 0.0)	(1.0, 0.0)
PCA (fixed, te, u)	(0.0, 0.25)	(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, te)	(0.0, 0.25)	(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., oe, u)		(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., oe)		(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., ohe, u)		(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., ohe)		(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., te, u)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 1.0)
Wilc. (fixed, oe, u)		(0.0, 1.0)	(0.0, 0.25)
Wilc. (fixed, oe)		(0.0, 1.0)	(0.0, 0.25)
Wilc. (fixed, ohe, u)		(0.0, 1.0)	(0.0, 1.0)
Wilc. (fixed, ohe)		(0.0, 1.0)	(0.0, 1.0)
Wilc. (fixed, te, u)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 0.25)
Wilc. (fixed, te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 0.25)
SCD (unidir., oe, u)		(0.0, 1.0)	(0.0, 1.0)
SCD (unidir., oe)		(0.0, 1.0)	(0.0, 1.0)
SCD (unidir., ohe, u)		(0.0, 1.0)	(0.0, 0.25)
SCD (unidir., ohe)		(0.0, 1.0)	(0.0, 1.0)
SCD (unidir., te, u)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)
SCD (unidir., te)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)
SCD (bidir., te, u)		(0.0, 1.0)	(0.0, 0.75)
SCD (bidir., te)		(0.0, 1.0)	(0.0, 0.75)

Table 3: Abrupt drift detection performance on the synthetic datasets, as False Positive Rate and Latency (FPR_s, L). Unidirectional SCD uses PCA preprocessing on AGRAW with one-hot and target encoders.

Method	SEA	Agraw1	Agraw2
PCA (cont., ohe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (cont., ohe)		(1.0, 0.0)	(1.0, 0.25)
PCA (cont., te)	(0.0, 0.25)	(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, ohe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, ohe)		(1.0, 0.0)	(1.0, 0.0)
PCA (fixed, te)	(0.0, 0.25)	(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., oe)		(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 1.0)
Wilc. (fixed, oe)		(0.0, 1.0)	(0.0, 0.25)
Wilc. (fixed, te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 0.25)
SCD (unidir., te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 1.0)
SCD (bidir., te)		(0.0, 1.0)	(0.0, 0.75)

Table 4: 500 rows wide gradual drift detection performance on the synthetic datasets, as False Positive Rate and Latency (FPR_s, L).

Method	SEA	Agraw1	Agraw2
PCA (cont., ohe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (cont., ohe)		(1.0, 0.0)	(1.0, 0.25)
PCA (cont., te)	(0.0, 0.25)	(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, ohe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, ohe)		(1.0, 0.0)	(1.0, 0.0)
PCA (fixed, te)	(0.0, 0.25)	(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., oe)		(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 1.0)
Wilc. (fixed, oe)		(0.0, 1.0)	(0.0, 0.25)
Wilc. (fixed, te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 0.25)
SCD (unidir., te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 1.0)
SCD (bidir., te)		(0.0, 1.0)	(0.0, 1.0)

Table 5: 1000 rows wide gradual drift detection performance on the synthetic datasets, measured in False Positive Rate and Latency (FPR_s, L).

Method	SEA	Agraw1	Agraw2
PCA (cont., ohe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (cont., ohe)		(1.0, 0.0)	(1.0, 0.0)
PCA (cont., te)	(0.0, 0.25)	(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, ohe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, ohe)		(0.5, 0.0)	(1.0, 0.0)
PCA (fixed, te)	(0.0, 0.25)	(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., oe)		(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 1.0)
Wilc. (fixed, oe)		(0.0, 1.0)	(0.0, 0.25)
Wilc. (fixed, te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 0.25)
SCD (unidir., te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 1.0)
SCD (bidir., te)		(0.0, 1.0)	(0.0, 0.75)

Table 6: 5000 rows wide gradual drift detection performance on the synthetic datasets, measured in False Positive Rate and Latency (FPR_s, L).

Method	SEA	Agraw1	Agraw2
PCA (cont., ohe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (cont., ohe)		(1.0, 0.0)	(1.0, 0.25)
PCA (cont., te)	(0.0, 1.0)	(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, ohe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, ohe)		(1.0, 0.0)	(1.0, 0.0)
PCA (fixed, te)	(0.0, 1.0)	(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., oe)		(0.0, 0.25)	(0.0, 1.0)
Wilc. (cont., te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 0.25)
Wilc. (fixed, oe)		(0.0, 0.75)	(0.0, 0.25)
Wilc. (fixed, te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 0.25)
SCD (unidir., te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 1.0)
SCD (bidir., te)		(0.0, 1.0)	(0.0, 1.0)

Table 7: 10000 rows wide gradual drift detection performance on the synthetic datasets, measured in False Positive Rate and Latency (FPR_s, L).

Method	SEA	Agraw1	Agraw2
PCA (cont., ohe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (cont., ohe)		(1.0, 0.0)	(1.0, 0.0)
PCA (cont., te)	(0.0, 1.0)	(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, ohe, u)		(0.0, 1.0)	(0.0, 1.0)
PCA (fixed, ohe)		(1.0, 0.0)	(1.0, 0.0)
PCA (fixed, te)	(0.0, 1.0)	(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., oe)		(0.0, 1.0)	(0.0, 1.0)
Wilc. (cont., te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 1.0)
Wilc. (fixed, oe)		(0.0, 1.0)	(0.0, 0.5)
Wilc. (fixed, te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 0.5)
SCD (unidir., te)	(0.0, 0.0)	(0.0, 1.0)	(0.0, 1.0)
SCD (bidir., te)		(0.0, 1.0)	(0.0, 1.0)

Table 8: 20000 rows wide gradual drift detection performance on the synthetic datasets, measured in False Positive Rate and Latency (FPR_s, L).

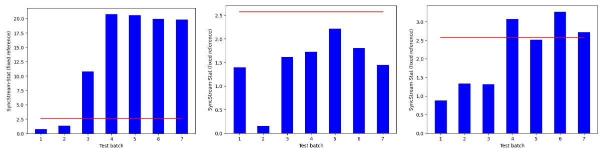

Figure 3: Example of difference in the Wilcoxon test results between abruptly drifting SEA and AGRAW datasets using a fixed reference, TargetEncoder and MinMaxScaler. On the left is the result on SEA. The test statistic raises noticeably above the critical value immediately and no false positives are reported. In the middle is the result on AGRAW1. The test statistic stays high after the drift, but is not significant and not distinguishable from the non-drifting batch 1. On the right, the test manages to detect the drift in AGRAW2, albeit late.

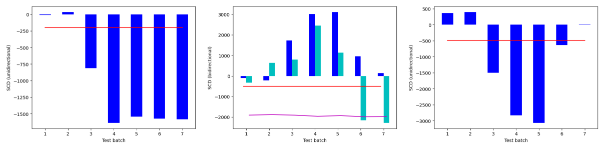

Figure 4: Illustration of the SCD results on abruptly drifting synthetic datasets. On the left is SEA (scaled), where it works as expected. In the middle is a bidirectional run on AGRAW2 (TargetEncoder, unscaled) where the test statistic grows in the wrong direction. The teal-colored bars and the purple line represent the reverse tests. On the right is the same AGRAW2 dataset with the same setup, but unidirectional and with PCA preprocessing added. The test statistic now behaves expectedly and the drift is detected correctly.

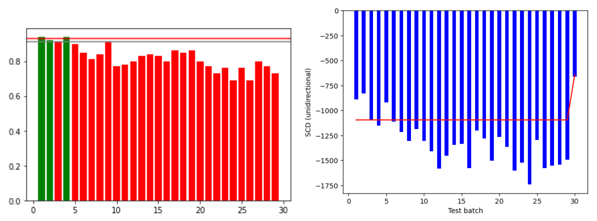

Figure 5: Similarities between the model accuracy and the inverted SCD test statistic on the Spam dataset with batch size 100. On the left, accuracies from the model used to define the drifts; on the right, the inverted test statistic follows the same general trend.

test manages to detect the drift in AGRAW2, one batch late, when using the fixed reference adaptation and some other encoder than one-hot. Overall, in a lot of cases, the Wilcoxon test does produce some signal for the drift, but it fails to be significant - see Fig. 3 for an example.

SCD exhibits some unexpected behavior on the AGRAW datasets as the test statistic grows in the positive direction when examining the drifting batches. Bidirectional mode slightly increased the performance by bringing the test statistic back to zero and into negative values faster - it still detects the drift considerably late. Finally, introducing PCA preprocessing - initially to fix the linear algebra errors from one-hot encoding - made the test statistic behave expectedly. See Fig. 4 for an illustration of the SCD results.

Gradual drift

Tables 4, 5, 6, 7, and 8 show the results on gradually drifting synthetic datasets of widths 500, 1000, 5000, 10000, and 20000, respectively. Most results for the gradual drifts follow the same pattern as for the abrupt drift. With drifts of 10000 and 20000 rows, we see that the PCA method stops working. SCD and Wilcoxon test both detect all drifts in SEA perfectly.

4.2 Real-world data

Results for the real-world datasets are shown in Tables 9, 10, 11, and 12 for Airlines, Elect2, Weather, and Spam, respectively. They differ significantly from those on the synthetic data. As a highlight, the Wilcoxon test had a zero false positive rate while also having an accuracy of 0.76 on Airlines with TargetEncoder.

False positive rate increased for SCD when run in bidirectional mode on the Weather dataset, although running in two directions was not supposed to do this [4]. This does not seem to be because of the PCA preprocessing; the effect was to be seen already before that was introduced.

On the other hand, with the PCA preprocessing down to 100 components, SCD displays the same kind of inverted behavior on Spam than it does without the preprocessing on AGRAW. Averages of 10 runs showed that by flipping the sign on the test statistic, SCD could detect the majority of drifts in Spam with batch size 100 while keeping the false positive rate well under 0.5. The results are included as interesting findings in Table 12. See also Fig. 5 for visualization.

5 Discussion

5.1 Synthetic data

Performance under abrupt drift

The PCA method performed rather poorly, but was very fast to run and thus the easiest to experiment with. On the AGRAW datasets, the measured eigenvector angles do not suggest any signal that could be found by improving the method. Since the method required adapting the threshold value to even work with SEA, its usability is questionable.

While performing decently on SEA, the behavior of SCD on the AGRAW datasets was unexpected; the test statistic is supposed to grow in the negative direction when the detector encounters a drift, as indeed happens with the SEA dataset [4]. It seemed as if the detection performance of SCD

Method	Airlines
PCA (fixed, oe, u)	(0.0, 0.0)
PCA (fixed, ohe, u)	(0.0, 0.0)
PCA (fixed, te, u)	(0.0, 0.0)
PCA (fixed, oe)	(1.0, 0.29)
PCA (fixed, ohe)	(0.0, 0.095)
PCA (fixed, te)	(0.0, 0.048)
Wilc. (fixed, oe, u)	(1.0, 0.67)
Wilc. (fixed, ohe, u)	(0.0, 0.67)
Wilc. (fixed, te, u)	(0.0, 0.76)
Wilc. (fixed, oe)	(1.0, 0.67)
Wilc. (fixed, ohe)	(0.0, 0.67)
Wilc. (fixed, te)	(0.0, 0.76)

Table 9: Drift detection performance on the Airlines dataset, measured in False Positive Rate and Accuracy (FPR_{rw}, Acc).

Method	Elect2
PCA (fixed, u)	(0.48, 0.54)
PCA (fixed)	(1.0, 1.0)
Wilc. (fixed, u)	(1.0, 0.93)
Wilc. (fixed)	(1.0, 0.71)
SCD (unidir., u)	(1.0, 1.0)
SCD (unidir.)	(1.0, 1.0)

Table 10: Drift detection performance on the Elect2 dataset, measured in False Positive Rate and Accuracy (FPR_{rw}, Acc). SCD uses PCA for preprocessing.

Method	Weather (monthly)	Weather (yearly)
PCA (fixed, u)	(0.95, 0.97)	(0.67, 0.56)
PCA (fixed)	(0.90, 0.96)	(0.27, 0.056)
Wilc. (fixed, u)	(0.84, 0.87)	(0.53, 0.61)
Wilc. (fixed)	(0.84, 0.87)	(0.53, 0.61)
SCD (unidir., u)	(0.18, 0.18)	(0.37, 0.51)
SCD (unidir.)	(0.16, 0.16)	(0.37, 0.52)
SCD (bidir., u)	(0.99, 1.0)	(0.80, 0.83)
SCD (bidir.)	(0.99, 1.0)	(1.0, 0.94)

Table 11: Drift detection performance on the Weather dataset, measured in False Positive Rate and Accuracy (FPR_{rw}, Acc). Monthly and yearly batching was used. Unidirectional SCD is averaged over 10 runs. Bidirectional SCD uses PCA for preprocessing.

Method	Spam (100)	Spam (50)	Spam (20)
PCA (fixed, u)	(1.0, 1.0)	(1.0, 1.0)	(1.0, 1.0)
PCA (fixed)	(1.0, 1.0)	(1.0, 1.0)	(1.0, 1.0)
Wilc. (fixed, u)	(1.0, 1.0)	(1.0, 0.98)	(0.88, 0.96)
Wilc. (fixed)	(1.0, 1.0)	(1.0, 0.98)	(0.88, 0.96)
SCD (unidir., u)	(.20, .63)	(.22, .45)	(.0022, .039)
SCD (unidir.)	(.34, .69)	(.16, .35)	(.0, .049)

Table 12: Drift detection performance on the Spam dataset, measured in False Positive Rate and Accuracy (FPR_{rw}, Acc). Different batch sizes were used. SCD uses PCA preprocessing with 100 components and inverted test statistic and is averaged over 10 runs.

would be notably improved for the AGRAW datasets if the sign of the test statistic was flipped. However, when fixing the one-hot encoding with the PCA preprocessing method, the test statistic actually did function as expected. This lead to an experiment with the PCA preprocessing and TargetEncoder, and that indeed works: with this setup, SCD is able to detect both AGRAW drifts perfectly, making it the only detector in this study to do so in all three synthetic datasets.

It was expected that these methods could be slow [2]. As noted before, SCD is indeed very expensive to run; it does not scale well when the training set is any larger than a few thousand rows. With our implementation and using a recent laptop, one unidirectional pass over a one-hot encoded and PCA-preprocessed AGRAW dataset averaged 42 minutes. Target encoding reduced that to 11 minutes, which seems to directly correlate with the number of columns. However, the SyncStream methods were still faster by an order of magnitude.

Performance under gradual drift

None of the detectors displayed significantly different performance on gradual drift compared to abrupt drift, the only exception being PCA when encountering drifts of 10000 rows and wider. In short, if a detector worked well on the abrupt version of a dataset, it worked on the gradual versions too.

The Wilcoxon test displayed better performance with the fixed reference adaptation on the AGRAW2 dataset. Part of the reason for this seems to be that it failed to meet the significance criteria whenever the change was too subtle, so when the reference batch contained some already-drifting data. This was the case in all our experiments on synthetic datasets, as the drifts always started in the middle of the batch.

The performance of SCD on AGRAW improved considerably with abrupt drift when the PCA preprocessing method was introduced. However, due to time constraints this was not tried on the gradual drifts.

5.2 Performance on real-world data

The detectors did not perform well on the real-world datasets. In addition to some setups refusing to run at all due to issues with data, we saw only rare occasions of FPR_{rw} of under 0.5 combined with Acc of over 0.5, meaning the detector would consistently beat flipping a coin.

The SyncStream algorithm originally achieved accuracy scores of over 0.8 when evaluated on a Spam dataset and an Electricity dataset [5]. However, this measured the performance of the full classification algorithm, and we took the detection methods out of that context, so the results are not comparable as such. It is worth noticing that a lot of the false alarms from the PCA method had almost 180 degree angle differences. This suggests that the principal eigenvector probably just flipped its direction, which is not necessarily an indication of a significant shift in the data.

Like on the non-preprocessed AGRAW datasets, SCD starts to produce reasonable results on Spam if the test statistic is inverted. Again, this is not supposed to happen, but our experimental results show this tendency on more than one dataset, so this is something future research should address.

In the Elect2 dataset, day represents the day of week and is considered a numerical feature in this study. However, the relation between electricity price and day of week may not be ordinal in nature. For any future research with this dataset, we suggest trying to treat day as a categorical feature.

One important point to consider is how to define concept drift. We took an approach that is concerned with model accuracy, but concept drift can also be defined simply as change in the data distribution [2]. Although it often is the case, this is not strictly the same as decreasing quality of the attached model. In this survey paper, the authors observe that the distribution could change without affecting the model - called virtual drift - and the quality could also deteriorate without a change in the distribution - called actual drift. Furthermore, they conclude that in real-world data we should expect to find a combination of these. We note that it is therefore not too surprising that drift detection algorithms based on data distribution may not perform optimally in such scenarios since they can only capture the virtual component of the drift.

6 Conclusions and Future Work

The research question of this study was *"How well do data distribution based concept drift detectors identify concept drift in case of synthetic/real-world data"*. As concluded by Poenaru-Olaru et al., research on label-independent, such as data distribution based, concept drift detectors is currently hindered by the lack of available implementations of such detectors [1]. In this study we implemented drift detection methods from two studies: the PCA and Wilcoxon test techniques of SyncStream [5], and SCD, also called density test [4]. The performance of these detectors was evaluated on both synthetic and real-world data. We publish our implementations for future research to verify and build on.

Our results show that performance on synthetic datasets is no guarantee for performance on real-world data. This suggests that for real-world applications it is not recommended to develop these algorithms with synthetic datasets. It was also found that in addition to encoding and scaling, preprocessing the data with PCA can have a significant effect on the drift detection performance.

Concept drift in real-world data can in most cases be split into two components: virtual drift and actual drift [2]. Virtual drift means a drift in the data distribution without an effect on the decision boundary; thus leaving the attached model unaffected by the change. We should therefore expect a data distribution based detector to report a false positive on a virtual drift. Actual drift means a drift of the decision boundary, thus deteriorating the quality of the attached machine learning model without a change in the data distribution. This kind of drift would now go undetected by a data distribution based detector. Considering this, quantifying to what extent virtual drift and actual drift occur together in real-world data sources could be an interesting line of research.

Acknowledgements

This paper is an individual contribution to the research project *Detecting Concept Drift in Deployed Machine Learning Models* at Delft University of Technology, supervised by Lorena Poenaru-Olaru. A group of students worked on the same topic, each with a different category of detectors.

Responsible Research

General ethical rules for scientific research require that it should be conducted responsibly, which includes not only considering the rights and consent of all participants, but also making an effort for the contributions to be verifiable and reproducible. Maintaining good reproducibility in computer science research, preferably by publishing source code whenever possible, reduces unnecessary work in further research as well as helps in building a critical peer community.

New data is not collected in this study; however, with the goal of benchmarking several drift detectors on the same datasets, the context of this paper is centered around reproducibility. We specifically implement algorithms that were previously published only in pseudocode and publish our own code in our personal GitHub repositories to avoid losing access after graduating. The file format for publishing the implementations was chosen to be .ipynb, which is widely used in the machine learning field. These factors make it significantly easier for future studies to reproduce our work and build on top of it.

According to the FAIR principles, research objects should be made "Findable, Accessible, Interoperable and Reusable" by both humans and machines [12]. Good commenting, file and variable naming, and other documentation is all therefore important to consider when publishing code and datasets. However, we make the case that even ambiguously documented or incomplete code would in many cases be better than no code when it comes to verification and reproduction.

To counter publication bias, we paid attention to including any insignificant, negative, and suboptimal results. These were obtained e.g. by trying out different encoders and data partitioning methods.

References

[1] L. Poenaru-Olaru, L. Cruz, A. van Deursen, and J. S. Rellermeyer, "Are concept drift detectors reliable alarming systems? - a comparative study," in *7th Workshop on Real-time Stream Analytics, Stream Mining, CER/CEP & Stream Data Management in Big Data*, 2022.

[2] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2019.

[3] D. Kifer, S. Ben-David, and J. Gehrke, "Detecting change in data streams," in *Proceedings of the 30th VLDB Conference*, Very Large Data Base Endowment, 2004.

[4] X. Song, M. Wu, C. Jermaine, and S. Ranka, "Statistical change detection for multi-dimensional data," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, (New York, NY, USA), pp. 667–676, Association for Computing Machinery, 2007.

[5] J. Shao, Z. Ahmadi, and S. Kramer, "Prototype-based learning on concept-drifting data streams," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, (New York, NY, USA), pp. 412–421, Association for Computing Machinery, 2014.

[6] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[7] A. Liu, G. Zhang, and J. Lu, "Fuzzy time windowing for gradual concept drift adaptation," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–6, 2017.

[8] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence – SBIA 2004* (A. L. C. Bazzan and S. Labidi, eds.), (Berlin, Heidelberg), pp. 286–295, Springer Berlin Heidelberg, 2004.

[9] V. Losing, B. Hammer, and H. Wersing, "Knn classifier with self adjusting memory for heterogeneous concept drift," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 291–300, 2016.

[10] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Tracking recurring contexts using ensemble classifiers: an application to email filtering," *Knowledge and Information Systems*, vol. 22, pp. 371–391, Mar 2010.

[11] Y. Lyu, H. Li, M. Sayagh, Z. M. J. Jiang, and A. E. Hassan, "An empirical study of the impact of data splitting decisions on the performance of aiops solutions," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, jul 2021.

[12] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, "The fair guiding principles for scientific data management and stewardship," *Scientific Data*, vol. 3, p. 160018, Mar 2016.