Augur: A Step Towards Realistic Drift Detection in Production ML Systems

Grace A. Lewis, Sebastián Echeverría, Lena Pons, Jeffrey Chrabaszcz
{glewis,secheverria,lepons,jschrabaszcz}@sei.cmu.edu
Carnegie Mellon Software Engineering Institute
Pittsburgh, PA, USA

ABSTRACT

The inference quality of deployed machine learning (ML) models degrades over time due to differences between training and production data, typically referred to as *drift*. While large organizations rely on periodic training to evade drift, the reality is that not all organizations have the data and the resources required to do so. We propose a process for drift behavior analysis at model development time that determines the set of metrics and thresholds to monitor for runtime drift detection. Better understanding of how models will react to drift before they are deployed, combined with a mechanism for how to detect this drift in production, is an important aspect of Responsible AI. The toolset and experiments reported in this paper provide an initial demonstration of (1) drift behavior analysis as a part of the model development process, (2) metrics and thresholds that need to be monitored for drift detection in production, and (3) libraries for drift detection that can be embedded in production monitoring infrastructures.

CCS CONCEPTS

• **Software and its engineering → Software design engineering**; • **Computing methodologies → Machine learning**.

KEYWORDS

software engineering, machine learning, drift detection, model monitoring, responsible AI

ACM Reference Format:
Grace A. Lewis, Sebastián Echeverría, Lena Pons, Jeffrey Chrabaszcz. 2022. Augur: A Step Towards Realistic Drift Detection in Production ML Systems. In *Workshop on Software Engineering for Responsible AI (SE4RAI'22), May 19, 2022, Pittsburgh, PA, USA*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3526073.3527590

1 INTRODUCTION

After machine learning (ML) systems are deployed, their models need to be retrained to account for differences between training and production data, known as *drift*. These differences over time lead to inference degradation — negative changes in the quality of ML inferences — which eventually reduce the trustworthiness of systems. Ideally, inference degradation would be quickly and reliably identified in production ML systems, allowing appropriate action to be taken (*e.g.,* retraining, cautioning users, taking the capability offline). The state of the practice in industry is to do periodic retraining and model redeployment to *evade* inference degradation instead of *monitoring* for inference degradation. Without an analytic basis for defining the appropriate retraining interval, this frequent retraining strategy risks correcting for inference degradation too slowly (*i.e.,* bad inferences may be the basis for critical decisions) or redeploying models too frequently (*i.e.,* retraining when it was not necessary, unnecessarily consuming resources, and increasing the risk of system downtime due to redeployment errors).

Current research provides several means of monitoring for drift, including measuring changes in prior or posterior probability, anomaly detection, and novel class emergence [24][26]. Existing work also defines a large set of metrics for drift detection (see Sections 2.3 and 7). However, there are gaps in this research that limit its use in production ML systems. First, metrics focus on detecting when drift has occurred and not why it is occurring to determine if retraining is required. Second, validation of these metrics is typically done using synthetic datasets or benchmarks that are not representative of production data (see Section 7). Finally, limitations such as (1) context dependency, (2) inability to detect different forms of drift (*e.g.,* continuous, abrupt, reoccurring), and (3) requirement of human validation to detect, make it difficult to use these metrics in production systems for reliable and timely inference degradation.

The main goal of our work is to provide a mechanism for drift behavior analysis and informed monitoring of production ML systems, an important aspect of Responsible AI. Our main contributions are (1) a method for introducing realistic drift into datasets, (2) a sample set of empirically-validated metrics that are predictors of when a model's inference quality will degrade due to different types of data drift, and (3) an extensible toolset developed for conducting experiments that forms the basis for supporting contextual drift behavior analysis as part of model development; determining metrics and thresholds that need to be monitored in production that would indicate drift; and providing reusable modules/libraries that can be embedded into model monitoring infrastructures to support realistic drift detection in production ML systems.

This paper is structured as follows. Section 2 presents the experiment setup. Section 3 explains how drift was introduced into the dataset used in the experiments. Section 4 describes the toolset created to run the experiments, which is one of the main outcomes of this work. Experiment results are presented in Section 5. Limitations are discussed in Section 6, and related work in Section 7. Finally, Section 8 concludes the paper and presents next steps to advance this work. All datasets and experiment results are available in the replication package at https://github.com/cmu-sei/augur-results.

SE4RAI'22 , May 19, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9319-5/22/05.
https://doi.org/10.1145/3526073.3527590

Open source toolset code is available at https://github.com/cmu-sei/augur-code.

2 EXPERIMENT SETUP

To show the value of drift behavior analysis we needed to demonstrate that the process could produce a set of drift detection metrics and thresholds to detect different types of drift that the model under analysis could experience in production. Therefore, we first had to determine what metrics were better at detecting what types of drift, which in turn required defining: (1) a production-relevant dataset, (2) a trained model that performs some inference on the dataset, (3) a series of known distortions to the dataset (drift induction), and (4) a set of metrics to estimate drift (drift detection).

2.1 Dataset and Model Selection

The goal was to select a dataset that would be closer to data found in production settings, than what we had observed in the existing literature (Section 7). We selected an open source dataset of synthetic aperture radar (SAR) images containing icebergs that were part of a Kaggle competition.[1] This dataset was selected because image classification is a very common computer vision task in production ML systems. Because the dataset had been published as part of a Kaggle competition, we were able to select a highly-performing model from the submitted solutions, instead of developing our own. We were also able to produce a full set of ground truth labels for all of the observations in that dataset. Because the model performed so well on the original data, we used inferences from the model to generate labels for the published test data as well, allowing us to treat both the published training and test data as a single corpus.

2.2 Drift Induction

The next step was to identify how we were going to introduce drift in the SAR dataset. Drift needed to be of sufficient magnitude to be detected, but also realistic. At a high-level, drift is caused by changes in the feature space (*i.e.,* real drift) or changes in data distribution over time (*i.e.,* virtual drift) [15]. An option for inducing real drift into image data is to change the pixel space of the input images via changing contrast, blurring, color, etc. An option for inducing virtual drift for data in general is to change the distribution of observations in the input data. Even though both types of drift are relevant to the SAR dataset, we selected to induce virtual drift because (1) it was better aligned to the drift scenarios that we wanted to evaluate for iceberg data (*e.g.,* seasonality, temperature changes), and (2) the drift functions would generalize more readily to other time series data. The drift induction process induces virtual drift by changing the prevalence of iceberg detections in the drifted dataset, as described in Section 3.

Because we induced virtual drift, we expect no effect of our drift induction on the classification model discussed in Section 2.1. Therefore we needed to organize the SAR dataset into a time series and create a model of the expected rate of iceberg detections over time. We created a time series of the SAR dataset by summing the detections in every contiguous set of 50 images, which we refer to as a time interval. To be able to predict the distribution of expected icebergs per time interval, we fit a statistical model to the base dataset, using an ARIMA (3,1,3) model [7]. Drift detection therefore becomes an estimate of the difference between the time series model and a time series aggregation of the drifted dataset.

[1] https://www.kaggle.com/c/statoil-iceberg-classifier-challenge/

2.3 Candidate Drift Detection Metrics

A review of the literature on drift detection identified approximately 40 unique drift detection approaches, grouped into five classes: distance-based metrics, error-based metrics, compound approaches, adaptive methods, and active learning based methods.

Distance-based metrics measure the distance between probability distributions of two segments of data (*e.g.,* Hellinger Distance and Energy Distance) [12]. Error-based metrics are statistical tests that entail constructing an explicit null hypothesis that new data will not deviate from training data outside of some range, dependent on the allowed ratio of false alarm (*e.g., z* statistic and Kolmogorov-Smirnov (KS) statistic) [16]. Compound approaches expand on distance or error metrics by first transforming the parameter space, which improves robustness in some cases (*e.g.,* Hellinger distance on a Principal Component Analysis of both training and production data [13]). The last two classes of approaches — adaptive approaches and active learning — were considered out of scope for this work because they are not focused on detecting drift but rather on applying dynamic changes to the production model to account for data drift (*e.g.,* Hoeffding trees [9] or core-set selection [22]). Based on a detailed analysis of each of these types of metrics, we selected six metrics for evaluation: Energy Distance, Hellinger Distance, Total Variation Distance, Kullback-Leibler (KL) Divergence, *z* statistic, and Kolmogorov-Smirnov (KS) statistic.

All six metrics have been used in previous work on drift detection, but differ in their sensitivity to differences in the compared samples. Energy, Hellinger, and Total Variation distance are proper distance metrics. KL Divergence, while not a true distance, is ubiquitous because of its relation to information theory. The final two metrics are statistical tests that are sensitive to differences in either the mean or the mean and standard deviation, respectively. We believe that this set of metrics covers different assumptions about the underlying data and is also sensitive to different deviations between the trained and observed data, *e.g.,* a *z* statistic will be sensitive to the difference between the means of two samples, while the KS statistic takes into account both the mean and dispersion.

3 MODELING DRIFT

With respect to change rate, there are several taxonomies of types of data drift (*e.g.,* [25]) that roughly classify drift based on duration (sudden vs. extended), transition (gradual vs. incremental), and reoccurrence (reoccurring vs. non-reoccurring). Based on the nature of possible drift in the SAR iceberg dataset we defined eight *drift scenarios* — descriptions of a drift condition expressed in terms of the problem domain — for different grades of sudden, gradual, and reoccurring drift, as shown in Table 1. The process to implement each drift scenario is as follows.

Step 1 - Define bins: Create a bin for each classification result. As an example, for the SAR dataset and model there are only two bins: *iceberg detected* and *iceberg not detected.*

Step 2 - Sort samples into bins: Place samples into each bin based on their classification.

Step 3 - Define sample groups and sizes: For each sample group that represents a unit of analysis, configure the percentage of samples to extract from each bin (*i.e.,* prevalences). The idea is that different prevalences represent different types of drift. As an example, sudden drift can be represented by 65% of icebergs detected in Sample Group 1 followed by 40% of icebergs detected in Sample Group 2.

Step 4 - Sort samples into groups: Randomly obtain samples from the bins, and place them into each group based on defined prevalences.

Step 5 - Generate timestamps: For each sample in the new drifted dataset, generate a sequential timestamp.

The parameters for each of the drift scenarios are also shown in Table 1. *Total Samples* is the resulting size of the drifted dataset. *Number of Sample Groups* defines the numbers of groups in which to divide the samples, with the resulting group size shown in *Sample Group Size*. *Prevalences* defines the percentage of samples of *Iceberg Detected* to place in each group. Finally, *Repeat Prevalence Range* applies only to the Reoccurring Sudden scenarios in which prevalences are generated randomly after the initial three, following the indicated randomization parameters.

4 EXTENSIBLE DRIFT BEHAVIOR ANALYSIS TOOLSET

We developed a toolset with extensibility as the driving quality attribute to enable researchers and model developers to extend the capabilities of the toolset beyond the scope of this project, such as (1) adding new datasets and models; (2) adding new drift types and drift induction functions; (3) adding new drift detection metrics; and (4) configuring new experiments. The toolset is targeted at model developers interested in analyzing model behavior in response to drift. In addition, once analysis is complete, the tool components that implement the drift detection metrics that end up being the best predictors of inference degradation, can be included in the monitoring infrastructure for drift detection in production (via a library of drift detection metrics). The toolset has three main components: Trainer, Drifter and Predictor. All components are command line tools with very detailed configuration files that can be used to customize each component as needed.

4.1 Trainer

The Trainer shown in Figure 1 trains the model under analysis. For our experiments, it trains the neural-network model for iceberg detection described in Section 2.1, as well as the supplementary time-series model described in Section 2.2 that aggregates the data over time. Using the Trainer to train the neural-network model is optional, because it could be trained separately using any ML framework (*e.g.,* TensorFlow, PyTorch, scikit-learn). However, the advantage of using the Trainer is that the model is exported in the TensorFlow format that is used by the Drifter and the Predictor. The Trainer is needed to train the supplementary time-series model based on the aggregated data from the neural-network model, as it will be used by the Predictor tool. Small Python modules need to be created to describe the dataset and the neural-network model, simply extending existing base classes and modules. The Trainer uses a configuration file to indicate the modules and parameters used for training. Adding new models and datasets for analysis simply requires their implementation in new Python modules and including links to them to the configuration file.

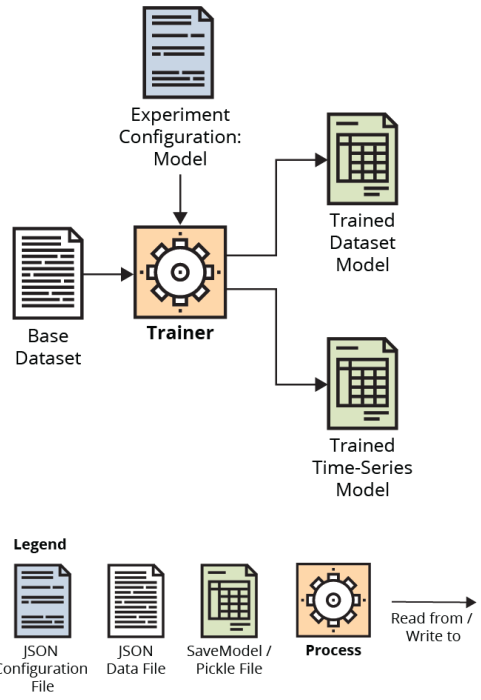

Figure 1: Dynamic View of the Trainer Component

4.2 Drifter

The Drifter shown in Figure 2 generates a drifted dataset by applying a drift induction function to a base dataset. The Drifter currently implements the prevalence drift function defined in Section 3, but additional functions can be added by creating simple Python modules that implement a defined interface for drift induction functions.

For the prevalence drift induction function, the output of the Drifter is a drifted dataset that contains samples from the base dataset, but in a different order to match the drift scenario, plus timestamps that extend the original data. A configuration file is used to adjust the parameters of the drift induction functions to create different types of simulated drift, as shown in the example in Figure 3. This file shows the configuration for the *Sudden* scenario from Table 1. Two `bins` are defined in Lines 5-9: 0 for *no iceberg detected* and 1 for *iceberg detected*. The `timestamps` to add to each sample in the drifted dataset are defined in Lines 11-14 (start on 2022-05-21 with increments of one hour). The drift scenario definition starts on Line 15: It implements the *Sudden* drift scenario (Line 17), using the drift induction function defined in the module `prevalence_drift`. The parameters for this module (following Table 1) are *Total Samples* in Line 21, *Sample Group Size* in Line 22, and *Prevalences* in Lines 23-32. This last parameter states that the *Prevalences of iceberg detected* (Bin 1) in each sample group, should be 65%, 40%, and 60%, with no repeat prevalence. Adding a new drift induction function simply requires its implementation in a new Python module (Line 18) and its parameters defined under the `params` tag (Line 19).

Table 1: Drift Scenarios and Parameters

Drift Type	Drift Condition	Total Samples	Number of Sample Groups	Sample Group Size	Prevalences (of *Iceberg Detected*)	Repeat Prevalence Range
Gradual	Decrease in number of icebergs with seasons	10026	6	1671	65, 63, 60, 55, 45, 40	N/A
Reoccurring Gradual	Reoccurring decrease in icebergs over multiple seasons	12000	12	1000	65, 55, 45, 30, 40, 50, 60, 50, 40, 30, 45, 55	N/A
Sudden	Sharp decrease in icebergs	9999	3	3333	65, 40, 60	N/A
Reoccurring Sudden	Multiple randomized sudden changes in the number of icebergs	12000	12	1000	65, 40, 60, ...	random [-5, +5]
Aggressive Gradual	More aggressive decrease in number of icebergs with seasons	10026	6	1671	65, 53, 40, 75, 35, 20	N/A
Aggressive Reoccurring Gradual	More aggressive reoccurring decrease in icebergs over multiple seasons	12000	12	1000	65, 45, 35, 20, 30, 60, 70, 60, 30, 20, 35, 45	N/A
Aggressive Sudden	More aggressive sharp decrease in icebergs	9999	3	3333	65, 10, 70	N/A
Aggressive Reoccurring Sudden	More aggressive multiple randomized sudden changes in the number of icebergs	12000	12	1000	65, 10, 70, ...	random [-10, +10]

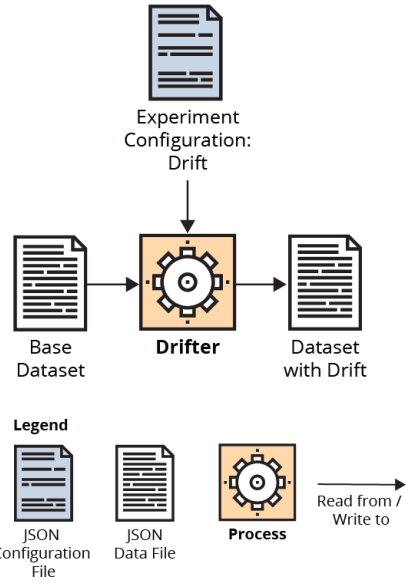

Figure 2: Dynamic View of the Drifter Component

```
5   "bins":
6     [
7       ["no_iceberg", 0],
8       ["iceberg", 1]
9     ],
10  "timestamps":
11  {
12    "start_datetime": "2022-05-21",
13    "increment_unit": "H"
14  },
15  "drift_scenario":
16  {
17    "condition": "sudden",
18    "module": "prevalence_drift",
19    "params":
20    {
21      "max_num_samples": 9999,
22      "sample_group_size": 3333,
23      "prevalences":
24      {
25        "1":
26        {
27          "percentages_by_sample_group": [65, 40, 60],
28          "prevalence_repeat": false
29        }
30      }
31    }
32  }
```

Figure 3: Example Configuration File for the Drifter

4.3 Predictor

The Predictor shown in Figure 4 runs the trained model with the drifted dataset generated by the Drifter and calculates the implemented drift detection metrics. The metrics defined in Section 2.3 are currently implemented in the tool, but others can be easily added by creating small Python modules that implement a defined interface for drift detection metrics. The Predictor also has a configuration file to configure different experiments that combine drifted datasets and metrics. Prediction results and metrics are generated as JSON files to be analyzed by model developers.

Figure 5 shows a snippet of a configuration file for the Predictor. Lines 17 to 21 configure the time interval for the time-series predictions, indicating where to start and what interval to use. The section starting on Line 22 defines the metrics that the Predictor will calculate on the data. Each item in the metrics array indicates one metric to be calculated by the Predictor. The first metric shown in Lines 24-34 is a distance metric, KL Divergence, and its parameters include the module that calculates it, and specific parameters to calculate density functions. The next two metrics (in lines 37 and 42) are error metrics, that simply define the name, type, and module used to calculate them. As noted earlier, adding new metrics simply requires creating a Python module that implements each metric.

5 EXPERIMENT RESULTS

The results of the experiments exist primarily in the *Prediction Metrics* file referenced in Figure 4, which is used for identifying the amount of inference degradation with respect to the induced drift at each time interval. The prediction metrics quantify the difference between the predicted number of icebergs (generated by the time-series model produced by the Trainer (Section 4.1)) and the number of observed icebergs in the drifted dataset in each post-training time interval. This output is summarized in Figure 6, which plots metric value (y-axis) by time period (x-axis) for each drift type

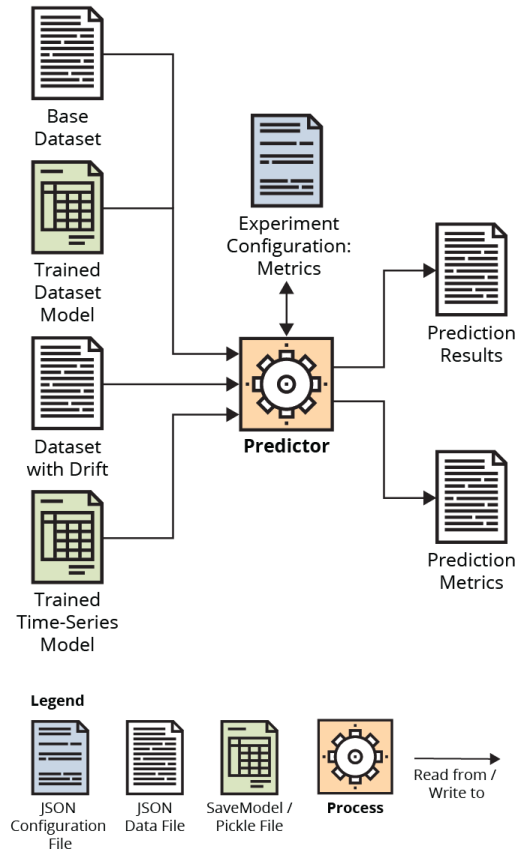

Figure 4: Dynamic View of the Predictor Component

```
"time_interval":
{
  "starting_interval": "2023-03-18",
  "interval_unit": "D"
},
"metrics":
[
  {
    "name": "kl-divergence",
    "type": "DistanceMetric",
    "module": "kl_divergence",
    "params":
    {
      "distribution": "normal",
      "range_start": -1,
      "range_end": 1000,
      "range_step": 1
    }
  },
  {
    "name": "Z Test / Student's t",
    "type": "ErrorMetric",
    "module": "z_test"
  },
  {
    "name": "Kolmogorov-Smirnov Statistic",
    "type": "ErrorMetric",
    "module": "kolmogorov_smirnov"
  }
]
```

Figure 5: Example Configuration File for the Predictor

Table 2: Metric Performance

Metric	Detection Threshold	Expected Time to Detection (Post-training Intervals)
Energy Distance	0.0001	1.3
Hellinger Distance	0.364	1.3
Kolmogorov-Smirnov Statistic	0.665	11
Kullback-Leibler Divergence	0.786	1.1
Total Variation Distance	0.0004	1.2
Z Test	-1.60	60

(line color) and metric type (facet). The plots were generated using ggplot2 (ggplot2.tidyverse.org) and CRAN R (R-project.org).

Table 2 shows the optimal threshold for drift detection in this dataset and the average time to detection for each metric, averaged across drift types. Between Table 2 and Figure 6 we can see that:

- Energy Distance detects drift quickly and is unique in that it increases monotonically as post-training time interval increases. All other tested metrics eventually drop below the detection threshold.
- Hellinger Distance generally detects drift in the first post-training time interval, but decreases over time as the prediction uncertainty grows.
- Kolmogorov-Smirnov Statistic is highly variable, taking an average of 11 time intervals to detect drift. This is likely because it is a one-sample test statistic calculated on a single observation. Like Energy Distance, though, this metric continues to probabilistically detect drift over the range of tested post-training intervals.
- KL Divergence shows immediate deflection that gradually returns to zero, but is the first metric to detect drift for all drift types in this dataset. This is a consequence of the prediction uncertainty on time-series models, which widens over time since the last training sample. One concern with using these metrics is that the system could look like it experienced temporary inference degradation and is slowly improving.
- Total Variation Distance shows a similar pattern to Hellinger Distance for all but the Aggressive Gradual drift, though the pattern is hidden by the large metric values associated with this drift type. Like Hellinger Distance, it is only slightly slower than KL Divergence at detecting drift.
- z Statistic suffers from the same instability as the Kolmogorov-Smirnov Statistic. Because of this, it is the last metric to detect any drift type in this dataset.

Based on these results and observations, we can make specific monitoring recommendations for the iceberg detection model. All drift types for this system can be adequately captured by KL Divergence and Energy Distance using the thresholds in Table 2. KL

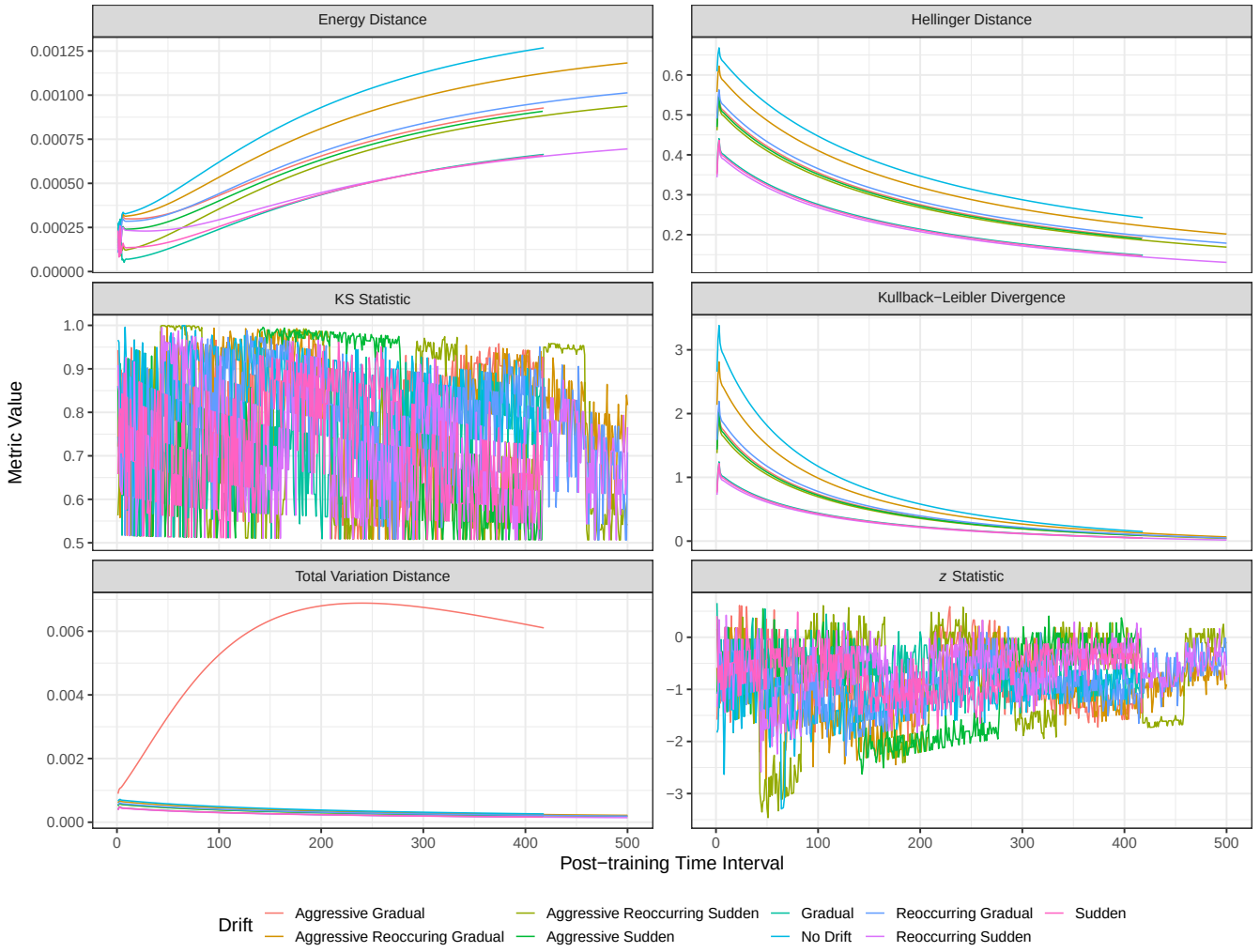

Figure 6: Metrics Values by Number of Time Intervals Post-Training, Colored by Drift Type and Faceted by Metric

Divergence shows a large value in the first post-training time interval for all drift types, but approaches zero as time since training elapses. Energy Distance also shows a small, immediate sensitivity to all drift types, but unlike KL Divergence, increases over time. Assuming these drifts are of the type and magnitude likely to be observed in production, KL Divergence gives an immediate signal that drift has occurred while Energy Distance provides accumulating evidence over time that the model's inferences have degraded.

These results demonstrate that we were able to detect different types of drift using a limited set of metrics. This supports the hypothesis that for production systems, an ensemble of drift detection metrics provides a monitoring capability that can simultaneously cover more types of drift. These results also suggest that, with sufficient planning prior to deployment, different patterns in metric change can suggest the nature of observed drift. This can be helpful to individuals responsible for maintaining models in production.

6 LIMITATIONS

While promising and encouraging, the results reported in this paper are limited in terms of drift scenarios analyzed, drift induction method implemented, and selected metrics, which are all specific to the SAR dataset and model that we used. However, as shown in Section 4, the toolset is easily extensible to to other drift inductions methods and metrics.

The toolset currently has limited analysis support; the reports generated by the Predictor require manual analysis, which would not have been possible without a data scientist on the team. However, the decision to export results in JSON was so they could be imported by tools used by data scientists for statistical analysis. This is one of several areas of future work (Section 8).

Finally, analysis was done with respect to only accuracy and detection delay (*i.e.,* time between drift appearance and its detection). Lack of consideration of system metrics (*e.g.,* performance, throughput, resource consumption) during model development is cited by

many practitioners as a cause for problems once models are put into production [17][21]. Providing additional metrics for analysis that include system metrics is an area of future work (Section 8).

7 RELATED WORK

The replication package contains an annotated bibliography that supports the statements and observations made in this section.

7.1 Drift Definitions and Types

There is a considerable amount of work in defining and characterizing drift associated to ML models, even if there is a lack of agreement on the terminology used (*e.g.,* drift vs. shift). Drift in general is defined as a change in model performance between training and production. In general, drift definitions fall into three categories [1][12][18][23][25]:

(1) *Covariate drift:* Differences in distribution between training data and production data caused by, for example, adversarial inputs, training data that is not representative of the operational domain, changes in data acquisition modules, or non-stationary environments (also referred to as virtual drift).

(2) *Probability shift:* Differences in the distribution of the target variable caused by, for example, changes in a policy or business rule that result in re-categorization of members of a class.

(3) *Concept drift:* Changes in context (*e.g.,* user behaviors) that lead to changes in the target variable (also referred to as real drift).

Our experiments were specifically targeted at introducing and detecting virtual drift (*i.e.,* covariate drift) which mapped to realistic drift conditions for the selected SAR iceberg dataset (Section 2).

7.2 Methods for Introducing Drift

Most related work leverages existing drifted datasets and benchmarks, as opposed to generating drifted datasets. Similar to our work, these drifted datasets are used for evaluation of drift detection methods. The most commonly used datasets are Massive Online Analysis (MOA) and MOA Extensions, available at https://moa.cms.waikato.ac.nz/ and https://sites.google.com/site/moaextensions, respectively. While extremely useful for developers of drift detection methods to use as a benchmark, these are not fully representative of production data streams. There are existing drift generator algorithms that we analyzed for suitability, such as the ones developed by Webb *et al.* [25]. In this work, we specifically focused on introduction of virtual drift as defined in Section 2.2, and in particular in the context of realistic drift conditions.

7.3 Methods for Addressing Drift in ML Systems

These methods fall into three main categories:

(1) *Ensembles:* Ensemble methods use the weighted outputs of a set of models, rather than a single model, and adjust the weights so that a higher weight is given to models that are producing outputs that are closer to the decision boundary. In this way, the ensemble adjusts to the effect of drift rather than having to fully retrain a model due to drift (active approach) [16].

(2) *Drift Detection Methods (DDMs)*: These methods flag when drift occurs and trigger some action to be taken (passive approach).

(3) *DDM Ensembles:* These are sets of DDMs that run in parallel with the goal of detecting different types of drift (active or passive approach based on implementation) [14][20].

Our work leverages existing DDMs to evaluate their effectiveness in detecting different types of drift. A large amount of DDMs have been proposed in the literature in the past 15 or so years. Many are based on metrics and tests such as ADWIN [6], CUSUM [2], DDM [4], KL Divergence [26], KS statistic [8], Hellinger Distance [1], and PCA [13], to cite some examples. These are also commonly used by most surveys that compare methods (*e.g.,* [3][5][11][12][14][16][19][24]). The results reported by these surveys were leveraged in the identification of drift detection metrics in this study (Section 2.3).

The end goal of the Augur toolset is to determine the *DDM Ensemble* that contains the metrics and thresholds that are the best predictors of the different types of drift that are likely to occur in production. This goal is similar to that of the work proposed by Babüroğlu *et al.* [3] in which they mapped DDMs to classifiers with the goal of determining the best DDMs for each classifier.

Finally, a large gap in related work, that is addressed by our work, is the lack of focus on production systems and production-readiness of ML models. In particular, much work focuses on identifying detection methods independent of the problem context and provides limited guidance on how to tailor these monitoring capabilities to specific systems. Other than Zhou *et al.* [26], there is very limited guidance or examples of how DDMs work in the context of production ML systems. As ML capabilities are integrated into more systems and are used to address more problem areas, our toolset can help model developers and engineers deploy systems with more confidence that they can capture model degradation over time.

8 CONCLUSIONS AND NEXT STEPS

We presented a process and toolset for realistic drift detection in production ML systems that makes the following contributions to the development of production-ready ML models: (1) a method to introduce context-specific drift into training datasets, (2) support for analysis of model behavior in the face of drift, and (3) reusable modules that can be integrated into model monitoring infrastructures for runtime drift detection. Our vision is to support the workflows shown in Figure 7. During model development, developers can use the Augur toolset to analyze model behavior and determine the best set of metrics and thresholds for drift detection over time. As shown in the diagram, models are retrained and new drifted datasets are created until developers are satisfied with the results. The resulting trained model, drift metrics and thresholds are used during model integration, development and operations for runtime drift detection. Once drift is detected, the resulting action could be simply an alert, manual or automated model retraining, model replacement, or additional monitoring and log data analysis.

However, additional work to support model production-readiness is required, which is the focus of our current and future work and includes: (1) Research and development of additional drift induction functions and drift detection metrics to be included in the toolset for developers to select during model development and evaluation, (2) Additional analysis metrics such as false positive rate, false negative rate, memory consumption, and processing time, (3) Evaluation of whether generating both a warning and a drift threshold leads to

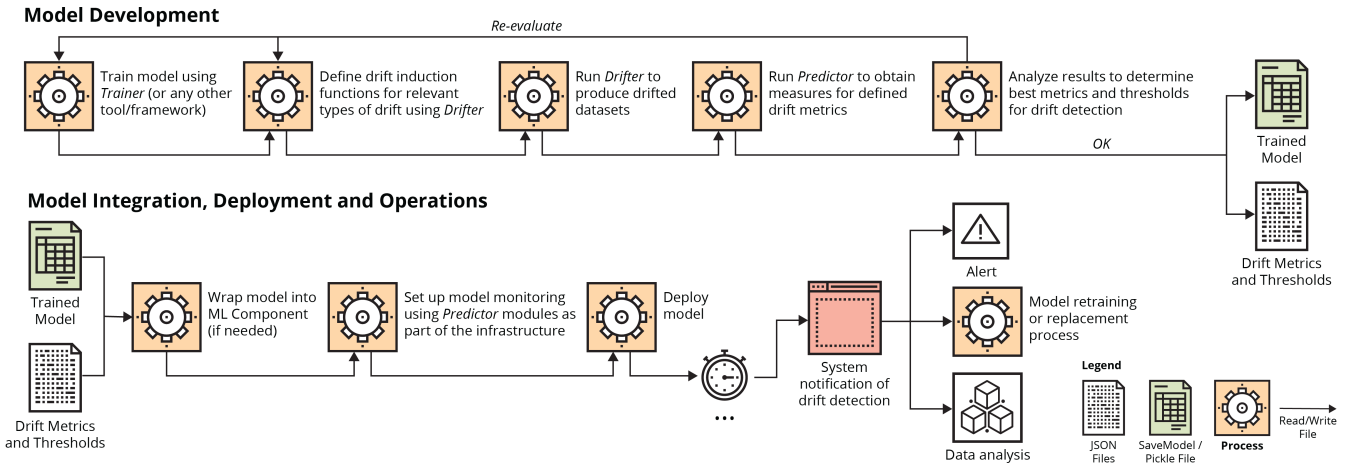

Figure 7: Workflows Supported by the Augur Toolset

the reduction of false positives [10], (4) Development and integration of a *Drift Analysis* component into the toolset that codifies the manual analysis that was done in the execution of this study, and (5) Generation of a code library for drift detection as an additional output of the model development workflow shown in Figure 7.

ACKNOWLEDGMENTS

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center (DM22-0229).

REFERENCES

[1] Hala Abdelkader. 2020. Towards Robust Production Machine Learning Systems: Managing Dataset Shift. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1164–1166.

[2] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri. 2016. Hierarchical Change-Detection Tests. *IEEE Transactions on Neural Networks and Learning Systems* 28, 2 (2016), 246–258.

[3] Elif Selen Babüroğlu, Alptekin Durmuşoğlu, and Türkay Dereli. 2021. Novel Hybrid Pair Recommendations based on a Large-Scale Comparative Study of Concept Drift Detection. *Expert Systems with Applications* 163 (2021), 113786.

[4] Roberto SM Barros, Danilo RL Cabral, Paulo M Gonçalves Jr, and Silas GTC Santos. 2017. RDDM: Reactive Drift Detection Method. *Expert Systems with Applications* 90 (2017), 344–355.

[5] Roberto Souto Maior Barros and Silas Garrido T Carvalho Santos. 2018. A Large-Scale Comparison of Concept Drift Detectors. *Information Sciences* 451 (2018), 348–370.

[6] Albert Bifet and Ricard Gavalda. 2007. Learning from Time-Changing Data with Adaptive Windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, 443–448.

[7] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons.

[8] David A Cieslak and Nitesh V Chawla. 2009. A Framework for Monitoring Classifiers' Performance: When and Why Failure Occurs? *Knowledge and Information Systems* 18, 1 (2009), 83–108.

[9] Pedro Domingos and Geoff Hulten. 2000. Mining High-Speed Data Streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 71–80.

[10] Isvani Frias-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jimenez, Rafael Morales-Bueno, Agustin Ortiz-Diaz, and Yaile Caballero-Mota. 2014. Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds. *IEEE Transactions on Knowledge and Data Engineering* 27, 3 (2014), 810–823.

[11] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A Survey on Concept Drift Adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 1–37.

[12] Igor Goldenberg and Geoffrey I Webb. 2019. Survey of Distance Measures for Quantifying Concept Drift and Shift in Numeric Data. *Knowledge and Information Systems* 60, 2 (2019), 591–615.

[13] Igor Goldenberg and Geoffrey I Webb. 2020. PCA-Based Drift and Shift Quantification Framework for Multidimensional Data. *Knowledge and Information Systems* 62, 7 (2020), 2835–2854.

[14] Paulo M Gonçalves Jr, Silas GT de Carvalho Santos, Roberto SM Barros, and Davi CL Vieira. 2014. A Comparative Study on Concept Drift Detectors. *Expert Systems with Applications* 41, 18 (2014), 8144–8156.

[15] Syed Muslim Jameel, Manzoor Ahmed Hashmani, Hitham Alhussain, Mobashar Rehman, and Arif Budiman. 2020. A critical review on adverse effects of concept drift over machine learning classification models. *International Journal of Advanced Computer Science and Applications (IJACSA)* 11, 1 (2020), 2020.

[16] Bartosz Krawczyk, Leandro L Minku, Joao Gama, Jerzy Stefanowski, and Michał Woźniak. 2017. Ensemble Learning for Data Stream Analysis: A Survey. *Information Fusion* 37 (2017), 132–156.

[17] Grace A. Lewis, Stephany Bellomo, and Ipek Ozkaya. 2021. Characterizing and Detecting Mismatch in ML-Enabled Systems. In *1st International Workshop on Software Engineering - AI Engineering (WAIN)*. IEEE.

[18] Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. 2012. A Unifying View on Dataset Shift in Classification. *Pattern Recognition* 45, 1 (2012), 521–530.

[19] Stephan Rabanser, Stephan Günnemann, and Zachary C Lipton. 2019. Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift. In *Advances in Neural Information Processing Systems (NIPS)*. 1396–1408.

[20] Jesse Read. 2018. Concept-Drifting Data Streams are Time Series; the Case for Continuous Adaptation. *arXiv preprint arXiv:1810.02266* (2018).

[21] Md Saidur Rahman, Emilio Rivera, Foutse Khomh, Yann-Gaël Guéhéneuc, and Bernd Lehnert. 2019. Machine Learning Software Engineering in Practice: An Industrial Case Study. *arXiv e-prints* (2019), arXiv–1906.

[22] Ozan Sener and Silvio Savarese. 2017. Active Learning for Convolutional Neural Networks: A Core-Set Approach. *arXiv preprint arXiv:1708.00489* (2017).

[23] Amos Storkey. 2009. When Training and Test Sets are Different: Characterizing Learning Transfer. *Dataset Shift in Machine Learning* 30 (2009), 3–28.

[24] Shuo Wang, Leandro L Minku, and Xin Yao. 2018. A Systematic Study of Online Class Imbalance Learning with Concept Drift. *IEEE Transactions on Neural Networks and Learning Systems* 29, 10 (2018), 4802–4821.

[25] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. 2016. Characterizing Concept Drift. *Data Mining and Knowledge Discovery* 30, 4 (2016), 964–994.

[26] Xianzhe Zhou, Wally Lo Faro, Xiaoying Zhang, and Ravi Santosh Arvapally. 2019. A Framework to Monitor Machine Learning Systems Using Concept Drift Detection. In *Intl. Conf. on Business Information Systems*. Springer, 218–231.