Carnegie Mellon University
Software Engineering Institute

Research Review 2021

Augur: Predicting Inference Degradation in Production ML Systems

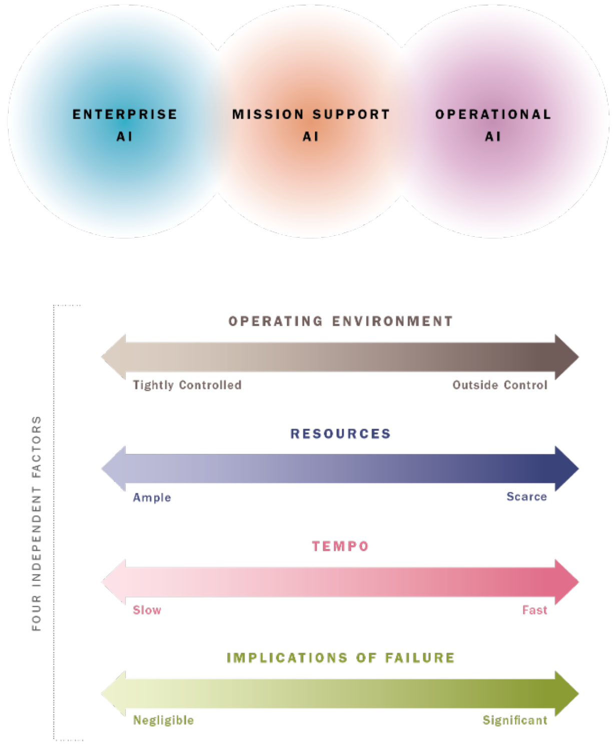
OCTOBER 4, 2021

Grace Lewis

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Motivation

The Spectrum of DoD AI Applications*

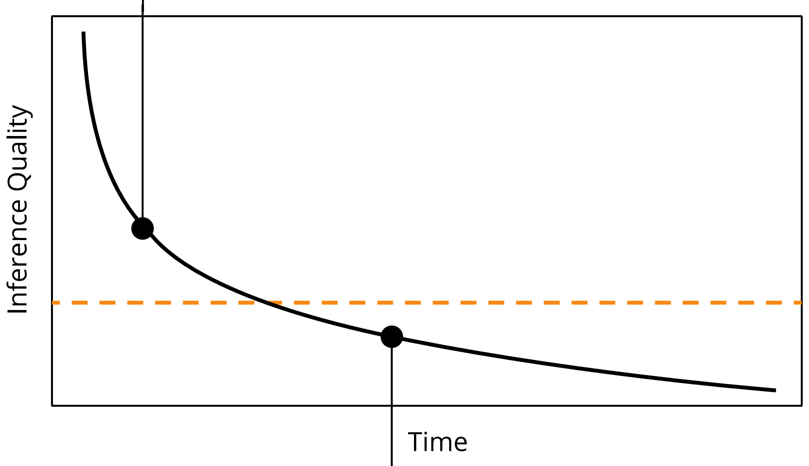


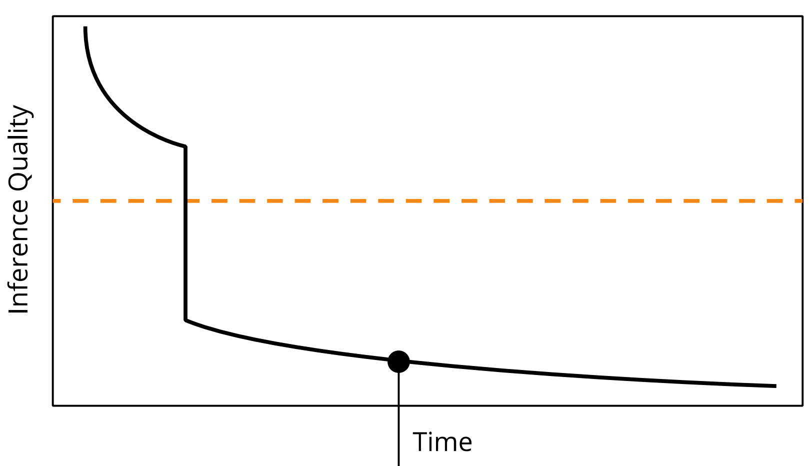
- Inference quality of deployed ML models changes over time due to differences between characteristics of training and production data
- State of the practice in industry relies on periodic retraining and model redeployment strategies to evade inference degradation
- Strategy of periodic retraining and redeployment becomes more infeasible as DoD AI systems move into the Operational AI space

* Reprinted with permission of RAND Corporation. Figure 3.2 “The Spectrum of DoD AI Applications,” from RR-4229-OSD, The Department of Defense Posture for Artificial Intelligence, ©RAND, 2019, https://www.rand.org/pubs/research_reports/RR4229.html. RAND is the original source and copyright holder.

Problem: Timely, Reliable Identification of Inference Degradation is Difficult

Too frequently: Wasting resources training and deploying retrained models when it was not necessary to do so, in addition to normal redeployment risks





Too late: A non-optimal course of action recommended by an ML capability may have already been put in place by the time the model is retrained and redeployed

Not all degradation is gradual: ML capability had to be taken offline while the model was retrained and redeployed

Failure to recognize inference degradation can lead to misinformed decisions, costly reengineering, and potential system decommission for DoD systems.

Solution

Carnegie Mellon University Software Engineering Institute

Develop a 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 therefore requires retraining

Develop a **test harness** that can help model developers understand model behavior due to drift and select appropriate monitoring metrics and thresholds

Metrics and thresholds can be integrated into MLOps pipelines (or ML model monitoring infrastructures) to determine

1. When a model really needs to be retrained
2. Thresholds that minimize the time that the model is producing sub-optimal results

Experiment Setup

1. Selected SAR as the DoD-relevant dataset for experimentation
   - Dataset from a Kaggle competition*
   - Goal was to correctly classify icebergs in satellite sea images containing other objects
   - Close to image classification datasets and data types used in DoD ISR applications
2. Identified relevant drift types to introduce in the SAR dataset
   - Needed to be of sufficient magnitude to be detected, but also realistic
   - Proved to one of the most challenging aspects of the project, as there is not much work in introducing **realistic drift** into datasets
3. Identified candidate metrics for identifying each type of introduced drift
   - Performed a literature review of drift detection methods to understand state of the research and practice

* https://www.kaggle.com/c/statoil-iceberg-classifier-challenge/

Experiment Set and Metrics

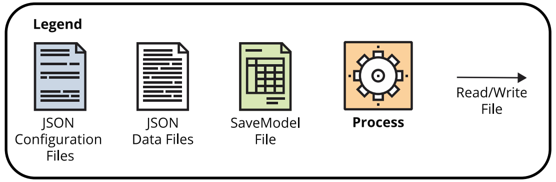
Carnegie Mellon University Software Engineering Institute

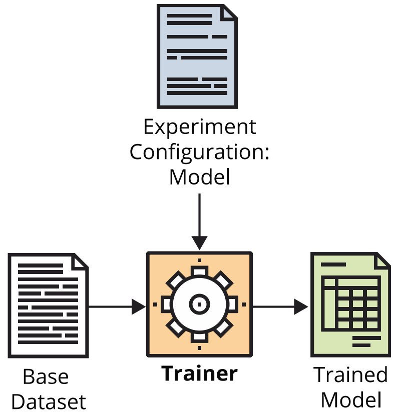
Scenario	Drift Types
Number of icebergs declines with seasons	Gradual (different rates) Gradual with false detection oversampling
Reoccurring decrease in icebergs over multiple seasons	Reoccurring gradual (different rates) Reoccurring gradual with false detection oversampling
Sharp decrease in icebergs is observed	Sudden (different rates) Sudden with false detection oversampling
Multiple randomized sudden changes in the number of detected icebergs	Reoccurring sudden (different rates) Reoccurring sudden with false detection oversampling

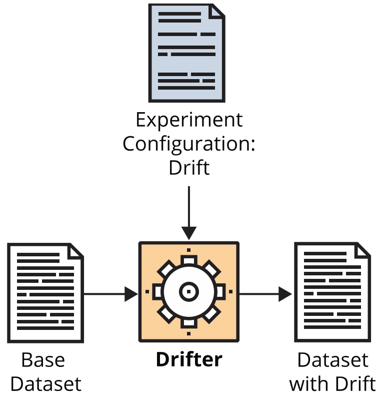
Metrics	
Distance Measure difference between two probability density functions	Kullback-Leibler (KL) Divergence
	Hellinger Distance
	Total Variation Distance
	Energy Distance
Error Compute a statistic that has a well-defined distribution corresponding to null hypothesis	Student's t test
	Kolmogorov-Smirnov (KS) Statistic

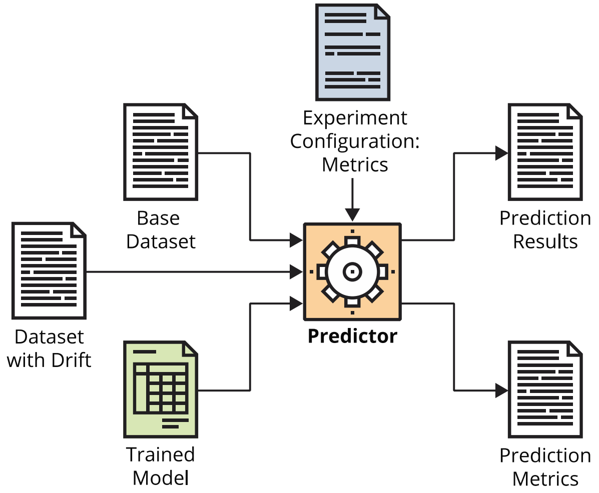
Test Harness

Composed of three tools







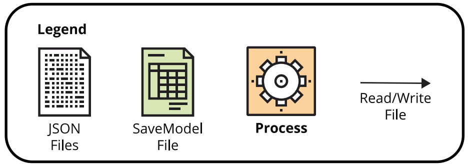


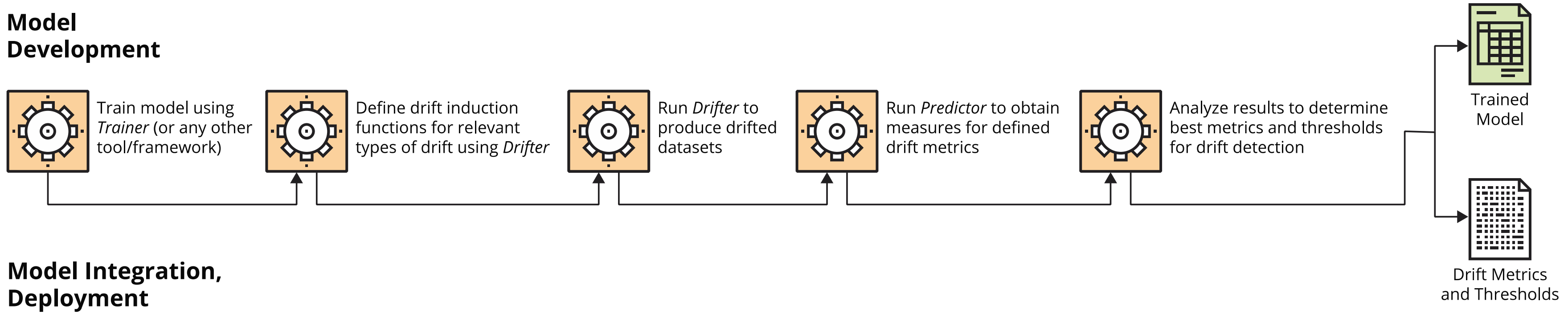
Trainer: Trains a CNN-based model with a given set of parameters — based on Tensorflow

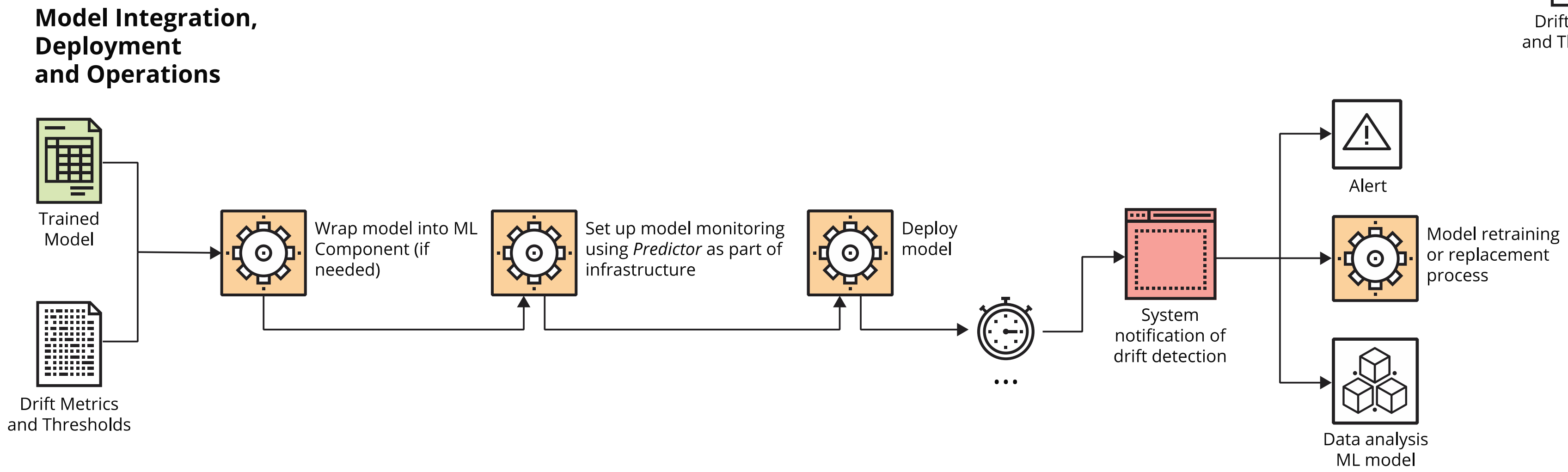
Drifter: Generates a drifted data set from a base data set and a drift type (python)

Predictor: Runs a drift detection experiment and generates metrics for analysis (python)

Supported Workflows




Model Development
Train model using *Trainer* (or any other tool/framework)
Define drift induction functions for relevant types of drift using *Drifter*
Run *Drifter* to produce drifted datasets
Run *Predictor* to obtain measures for defined drift metrics
Analyze results to determine best metrics and thresholds for drift detection
Trained Model
Drift Metrics and Thresholds


Model Integration, Deployment and Operations
Trained Model
Drift Metrics and Thresholds
Wrap model into ML Component (if needed)
Set up model monitoring using *Predictor* as part of infrastructure
Deploy model
...
System notification of drift detection
Alert
Model retraining or replacement process
Data analysis ML model

Extensibility and Flexibility of Test Harness

Carnegie Mellon University Software Engineering Institute

Architecture of test harness includes **extension points** for new

- Datasets
- Drift induction functions
- Drift detection metrics

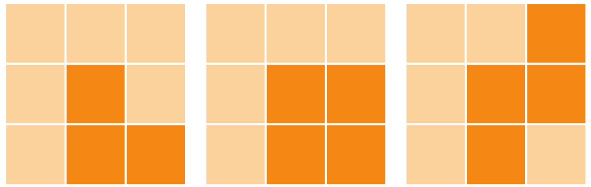
Definition of experiments using **configuration files** supports multiple uses of drift analysis

- Defining relevant drift metrics and thresholds for monitoring
- Evaluating model behavior with different drift types
- Deciding when to incur the cost of new labeled data for retraining

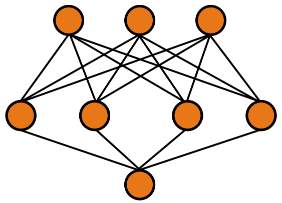
Modeling Drift Behavior

There are two options

1. Apply drift to feature space …



… and observe the drift independent of time

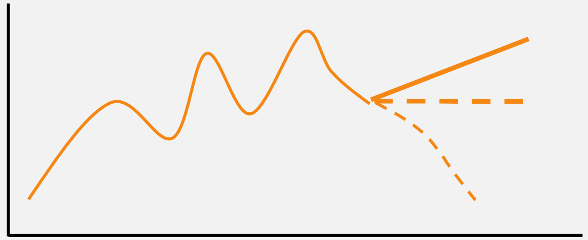


0.818 0.761 0.538

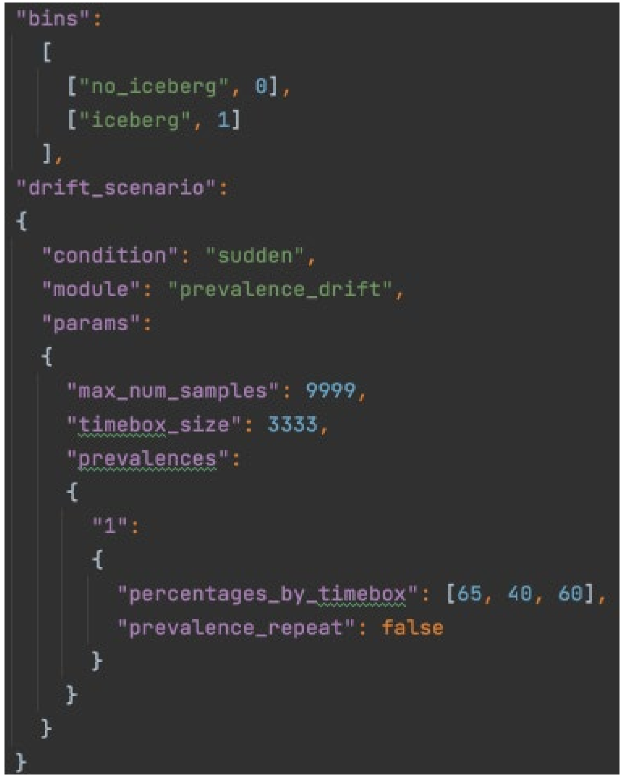
Selected the second option because it was more compatible with the scenarios that we wanted to evaluate in the SAR dataset

2. Apply drift to base rates …

… and observe the drift by time series



Drift Induction Process



```
"bins":
  [
    ["no_iceberg", 0],
    ["iceberg", 1]
  ],
"drift_scenario":
{
  "condition": "sudden",
  "module": "prevalence_drift",
  "params":
  {
    "max_num_samples": 9999,
    "timebox_size": 3333,
    "prevalences":
    {
      "1":
      {
        "percentages_by_timebox": [65, 40, 60],
        "prevalence_repeat": false
      }
    }
  }
}
```

1. Sort samples into configured bins by their classification
2. For each configured timebox
   a. Use the configured drift function to choose the next bin to get a sample from
   b. Get a sample from that bin, and add it to the timebox
   c. Repeat until there are all samples needed for this timebox

For step 2.a we implemented a “prevalence” type of drift

- For each timebox we configure what percentage of samples we want from each classification
- Drift function selects bins, keeping track of percentage for each classification
- Different percentages can be used to generate different types of drift (e.g., gradual, sudden, reoccurring)

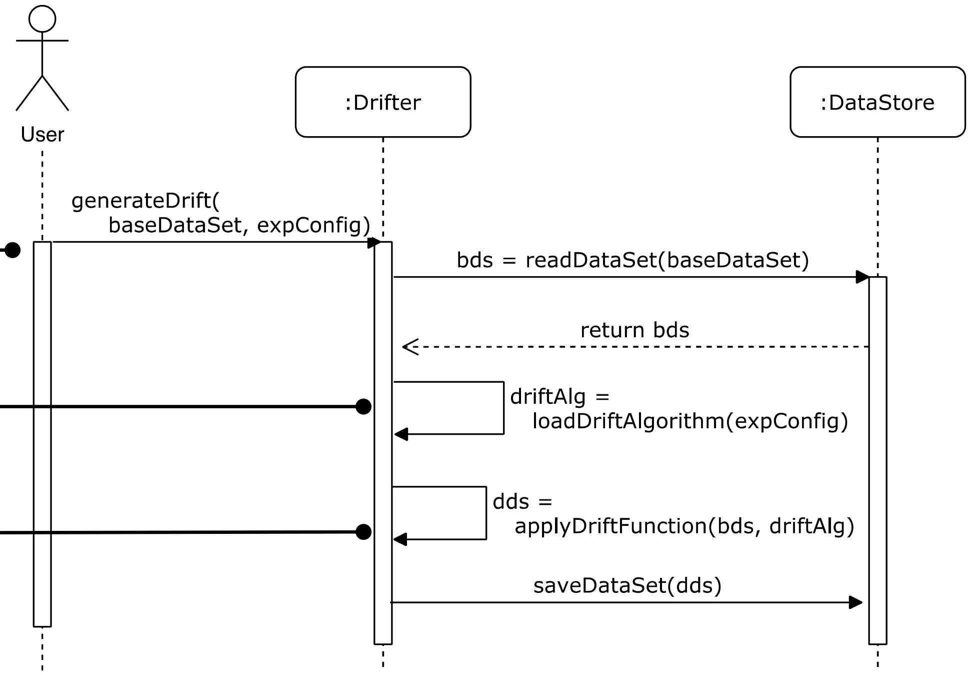
Drifter Details

Carnegie Mellon University Software Engineering Institute

Drifter is called with base dataset and experiment configuration details, including drift type to introduce

Drift algorithm matching drift type is loaded and applied to dataset

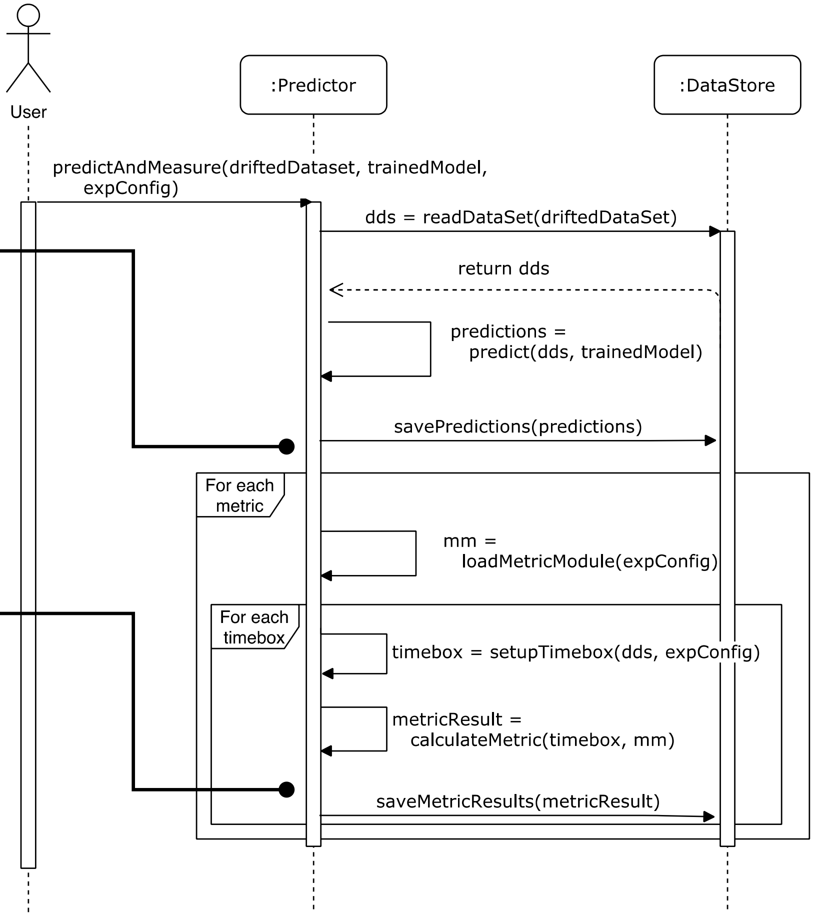
Dataset with drift, linked to base dataset, is ready for use by Predictor tool

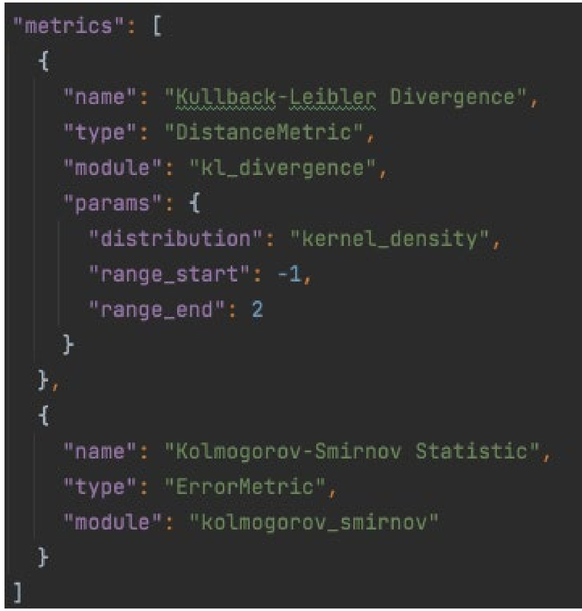


Predictor Details

Trained model is run with drifted dataset and predictions are saved in a Predictions file

Each configured metric is calculated for each timebox and saved in a Metrics file


User
:Predictor
:DataStore
predictAndMeasure(driftedDataset, trainedModel, expConfig)
dds = readDataSet(driftedDataSet)
return dds
predictions = predict(dds, trainedModel)
savePredictions(predictions)
For each metric
mm = loadMetricModule(expConfig)
For each timebox
timebox = setupTimebox(dds, expConfig)
metricResult = calculateMetric(timebox, mm)
saveMetricResults(metricResult)


```
"metrics": [
  {
    "name": "Kullback-Leibler Divergence",
    "type": "DistanceMetric",
    "module": "kl_divergence",
    "params": {
      "distribution": "kernel_density",
      "range_start": -1,
      "range_end": 2
    }
  },
  {
    "name": "Kolmogorov-Smirnov Statistic",
    "type": "ErrorMetric",
    "module": "kolmogorov_smirnov"
  }
]
```

Results from Initial Experiment Runs

Selected model was not sensitive to the way that we induced drift

Type of drift that we induced is more relevant to time-series models of counts of icebergs rather than a classification model to identify if an image contains an iceberg.

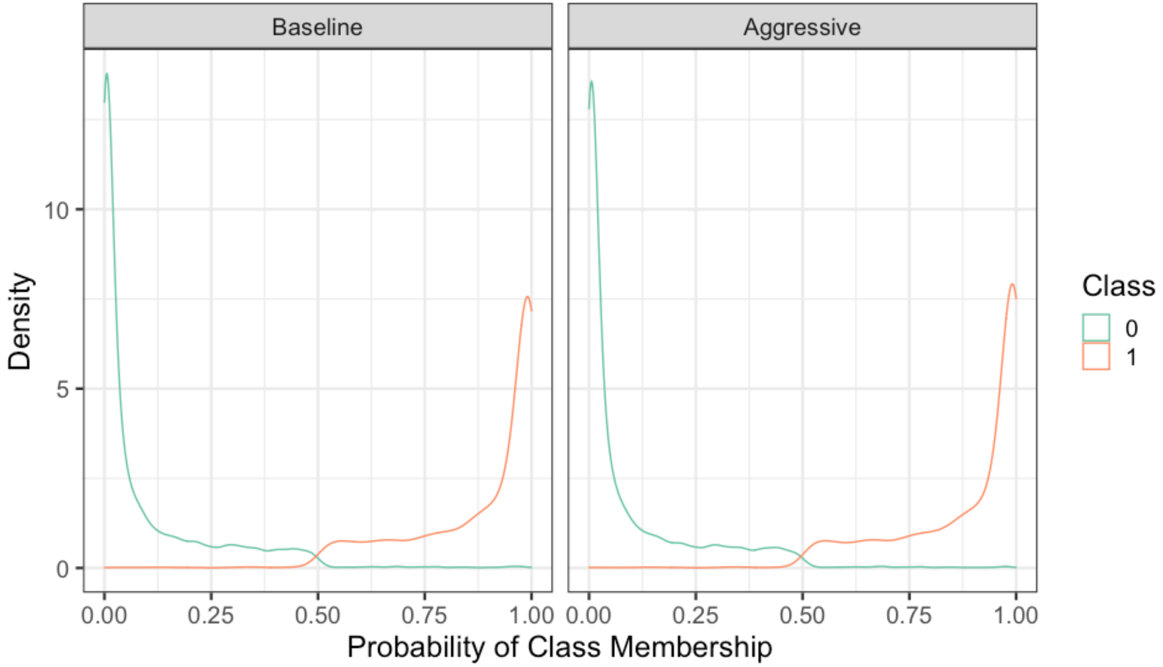


Diagram shows that our most aggressive drift scenario had limited impact on categorization model performance

Next Set of Experiments

Next step is to build a time-series model fit to the aggregated output of the original classifier

However, to test our hypothesis we constructed simulation experiments to establish expectations about the effect of drift on a time-series model

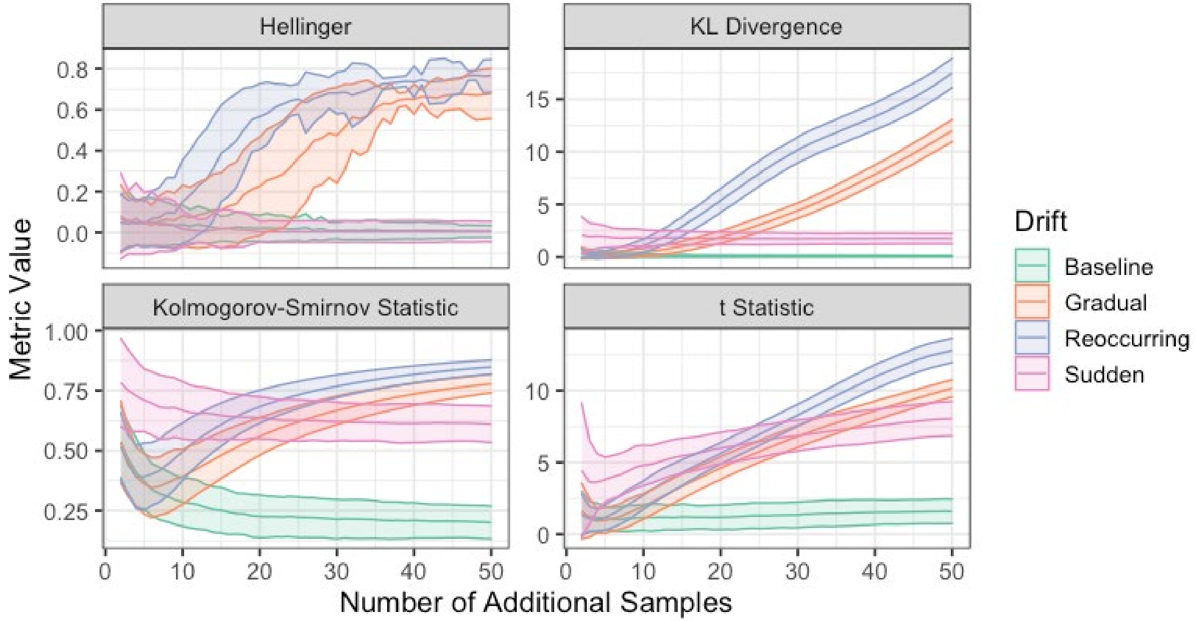
1. Generate a stationary, poisson-distributed time-series as baseline
2. Add deterministic drift and seasonality components consistent with the drift functions implemented in the test harness
3. Observe the effect of drift with known properties on a relevant subset of metrics

Results with Simulated Data Experiments

Carnegie Mellon University Software Engineering Institute

Drift induction detected with changes in metric values

- KL Divergence and the KS Statistic detect sudden drift quickly, but Hellinger does not detect it
- KL Divergence, KS Statistic and t statistic all show divergence between gradual, reoccurring and sudden drift within 10 new samples



Ribbons indicate the 68% uncertainty interval from 1000 experimental replications.

Project Contributions

Easy-to-use, extensible, and flexible **test harness software**

- Supports model development and model deployment/operation workflows
- Provides input to MLOps pipelines for what drift metrics and thresholds to monitor in production
- Supports new datasets, drift induction functions, and drift detection metrics via extension points built into the harness

Set of **validated metrics** for detecting different types of drift

Realistic drift induction functions that can be applied to other datasets

Summary of state-of-the art drift detection methods

Next Steps

Conduct additional experiments with a new time-series model

Document and publish results in both a software engineering and a machine learning venue

Release test harness as open source by end of December 2021

- Will help us collect feedback on DoD-relevant scenarios, drift types, and applications

Team

Grace Lewis (PI)

Lena Pons

Sebastián Echeverría

Jeffrey Chrabaszcz

Contact: info@sei.cmu.edu