PYTORCH ECOSYSTEM DAY

TorchDrift Drift Detection for PyTorch

A library making your PyTorch deployments more reliable

Thomas Viehmann, Luca Antiga, Daniele Cortinovis, Lisa Lozza
MathInf GmbH
Orobix srl

A1

Does your Deployed Model Work to Spec?

Before we deployed our model, we typically have validated it and obtained a measure of expected performance. But how can we be sure that the deployment environment still matches our validation scenario?

In a slight oversimplification, let us consider our model as a function $f : x \mapsto y$ that aims to maximize the conditional probability Prob(Y =y | X = x). Following S. Rabanser et al., we can define drift as an unexpected shift in the observed joint distribution P(X, Y) compared to some reference distribution $P_{ref}(X, Y)$ (e.g. that observed during validation or at initial deployment).

In drift detection, we aim to identify whether drift has happened, the perhaps most common example is that the distribution of X has shifted, i.e. the inputs have drifted. This is related to but distinct from domain adaptation, where we seek to mitigate the impact a shift in X on the prediction quality P(Y=f(x) | X).
In contrast to outlier detection, which detects whether individual samples are "unusual", we look at a batch of samples to determine whether they can be considered to be drawn from the same distribution on which we trained or validated our model.

Two Sample Testing to the Rescue

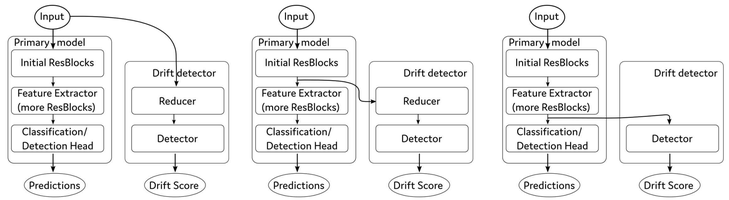
Bringing statistics to the problem

This formalization of drift detection points us to a tool from statistics that is mostly ready for our use: statistical two-sample tests.
More precisely, we perform statistical tests against the null hypothesis that no drift has happened, i.e. the distribution $P(X,Y) = P_{ref}(X, Y)$ is unchanged from the reference. In practical applications, we typically do not have the target Y, so we either only consider X or some value derived from this. This is tested based on a batch of samples of each the reference and the current distribution.

Some of the bread and butter tests – e.g. the Kolmogorov-Smirnov test – are one-dimensional, so to use them, we need to decide how to heuristically approximate our problem by a series of 1d tests. Other tests – e.g. the Kernel Maximum Mean Discrepancy test – are multidimensional from the start but work best in a small to moderate number of dimensions: As with anything statistics, the curse of dimensionality strikes here, too.
This means that depending on the "natural" dimension of our features, we may have to reduce dimensionality first, either using traditional methods like principal component analysis (PCA) or neural networks (parts of our model itself, autoencoders, ...).

Operationalizing Drift Detection

The first thing we have to decide on is what data to consider. Looking at a typical image classification model, we have a number of choices from taking the inputs the features to the class probabilities.



The drift detection pipeline – optional reducers and detectors – can be hooked up to various stages. The figure shows three typical setups (on input, on intermediate layers, on features).

Considerations here:

- Our model sure extracts interesting things from the images.
- But models tend to have blind spots and relying on model features might make us blind to some types of changes we want to catch.

Setting up a drift detector - a reducer and detecor can be combined in a nn.Sequential.

```
import torchdrift
detector = torchdrift.detectors.KernelMMDDriftDetector()
dl_train = torch.utils.data.DataLoader(ds_train, batch_size=N_train, shuffle=True)
feature_extractor = model[:-1]  # without the fc layer
torchdrift.utils.fit(dl_train, feature_extractor, detector, num_batches=1)
```

Deployment Considerations

Data for testing
The drift detection needs to balance timeliness of results (less data takes less time to sample) with detection power (more data makes it easier to spot drifts). One way to implement the pooling of data across several model invocations is to use a ring-buffer of samples that is then fed into the detector.

There are several options to integrate into the model deployment:

- We may use model hooks on the feature extractor and run drift detection from them. This can signal drift through callbacks or by raising exceptions or warnings.
- We may integrate the drift detector into the model as a submodule and return drift detection results along with those of principal interest.

We can hook the model monitor on the second-to-last layer of the model. It will accumulate test data, run the detection and pass detection score or p-value to a callback.

```
mm = ModelMonitor(detector, model[-2], callback=check_and_alarm)
```

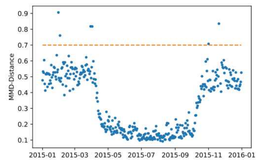
JIT
After calibration, TorchDrift's modules are plain PyTorch modules and can be traced to obtain TorchScript models. We are looking into scripting support.

End-to-End Example

We use a pre-trained TorchVision MaskRCNN to detect boats on the lake.
But we wish to check if our system is working as expected. We take a typical summer day as our reference.

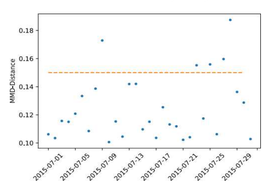
A normal day at the lake

We use days (approximately 120 pictures each) as test samples. We can see out reference point (which has distance 0). One general observation from real-life deployments is that sometimes the usual statistical tests tend to produce rather large deviations, so we need to calibrate the alarm threshold, e.g. by using a series of benign observations to estimate the distribution under the "no drift" null hypothesis which is empirically weaker than the null hypothesis of the two-sample-tests.



As the reference was a summer day, there is a strong seasonality. (Possibly also from the absence of boats in winter.) The days with the highest distances correspond to fog inhibiting normal operation. On the right are the pictures from 10am on the 6 days with distances above the 0.7 line.

We see that spring and summer days are generally closer than winter. This may be expected. The days that have extremely high distance from the reference correspond to foggy weather to the point of detection not working. Zooming in on July, we see that there, too, high MMD distances from the reference indicate unfavorable lighting conditions on cloudy days.



Left two picture columns: Noon on the four days with largest distances from the reference. In the right column, noon on two arbitrary other days.

For example notebooks with complete code, see our documentation at https://torchdrift.org/ or visit our breakout session.

References & Related Projects

S. Rabanser et al.: Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift, NeurIPS 2019

A library doing similar things for TensorFlow is A. Van Looveren et al: Alibi-Detect: Algorithms for outlier and adversarial instance detection, concept drift and metrics. (Retreived in February 2021)

Take a photo to learn more or visit https://torchdrift.org/

PyTorch