Why do ML Models Fail after Deployment?

Imagine you train and validate a machine learning (ML) model with high predictive accuracy and you have determined that it's ready for deployment. Once your model is deployed in the real world, you might think that this is cause for celebration as your hard work is now complete! However, the journey isn't over quite yet. When it comes to machine learning, you will need to continuously monitor your model's performance in the real world to ensure that it is still performing as well as it did when you validated its accuracy in production.

The reason ML models must be monitored after deployment is owing to the possibility of *model drift*, sometimes referred to as 'model decay', where the model degrades over time leading to a drop-off in predictive power. Why does model drift occur? Put simply, every model is created with the assumption that the way the world will be in the future is similar to how we interpreted the world at the time the model was developed. Of course, this assumption is not always warranted, as the real world is a dynamic environment that is constantly evolving.

When a model is deployed in an environment that changes in unforeseen ways, the model's performance is likely to change as well. In order to maintain the predictive accuracy of a model, successful deployment requires continuous monitoring over time to guard against model drift. Otherwise, the deployment of a model may lead to failure, with unhappy customers complaining that the model doesn't work as promised.

David Talby recounts his own experience with model drift in the article, "Why Machine Learning Models Crash and Burn in Production," in hopes that others may learn from his mistakes. After he deployed an ML model in hospitals to predict 30-day readmissions, he discovered a drop-off in performance after just 2-3 months. Moreover, the particular causes of model drift varied between hospitals, "or even buildings within the same hospital." For example, in some cases the problem was 'missing values' owing to changes in the electronic health record system. In other cases, the population of patients presenting to the ER had shifted after the hospital began accepting a new type of insurance.

Talby's experience is informative as it shows how a variety of factors can contribute to model drift. At this point, we will shift our attention to learning how to distinguish between the two most common causes of model drift: data drift and concept drift.

Data Drift

When data drift occurs, the properties of the data inputs are different from the data that was used to train the model. Similar to how Talby saw a change in the population of patients presenting to the ER, it's easy to imagine how the distribution of data may change with respect to TinyML applications.

Let's suppose you are creating a keyword spotting application to activate a device. Since the intended context for deployment consists almost entirely of native English speakers, you acquire data from this population to train your model. Fast forward to a few months after deployment--the model's accuracy has held steady at around 85%. Now let's suppose the context has suddenly changed, as the environment the device was deployed in has become a popular tourist destination. Since your model is encountering a large number of non-native English speakers with thick accents, the model's performance has dropped to 60%. This example shows how an unforeseen change in the real world led to a change in the distribution of data inputs, subsequently causing data drift.

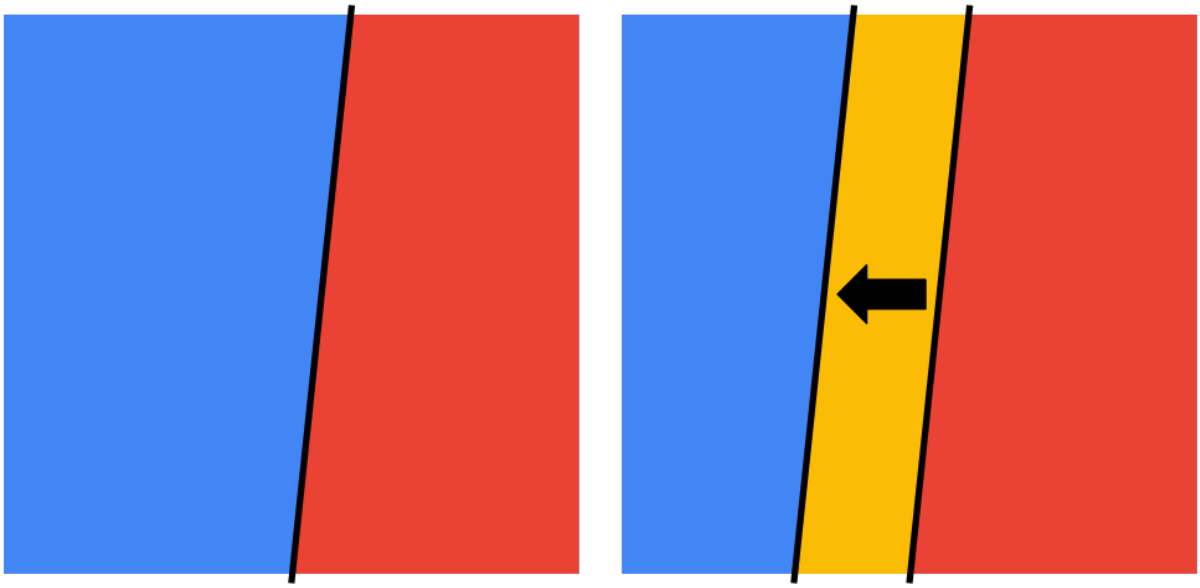
In addition to changes in the data inputs due to externalities, *upstream data changes* can also contribute to data drift. This is especially relevant to TinyML which exists at the intersection of machine learning and embedded IoT devices. On-device machine learning requires data inputs from sensors, and these sensors can raise a number of problems. For example, if a sensor is broken this might result in missing values which are required for making a prediction. Alternatively, the sensor itself may still be functioning properly but nevertheless result in faulty data inputs, e.g. if a camera lens is smudged and produces blurry images. Moreover, the sensor being used on the deployed device may be different than the sensor used to acquire the training data, and this can result in measurement changes, e.g. if the training data came from a high resolution camera but the model is deployed on a device with a low resolution camera.

Concept Drift

When concept drift occurs, the statistical properties of the target variable (which is what the model has been designed to predict) have changed. In other words, the actual relationship between the data inputs and outputs is different from what the model has learned. It is worth highlighting that concept drift can occur even when the distribution of incoming data remains the same and still accurately resembles the data used to train the model.

In Course 2, we discussed how defining the target variable is more like an art form than a perfect science, because it is a subjective process wherein one attempts to define some concept in quantitative measures e.g. defining 'healthy' in terms of measurable features like heart rate, number of steps, etc. Similar to how we might worry about how bias can influence the definition of the target variable, we also need to consider how the definition of the target variable might change or evolve over time in unforeseen ways.

For example, imagine we create a TinyML application to predict whether a plant is diseased by identifying features like yellow spotting and wilting leaves (see the left panel of the figure below). Now let's suppose that advances in research have led to a better understanding of this plant disease by identifying a new feature for accurately diagnosing it. As a result, some plants that would normally be classified as 'healthy' are now classified as 'diseased'. Since the concept of 'disease' has changed, our model's decision boundary is no longer accurate (see the right panel of the figure below). As a result, our model will incorrectly label data that falls between the decision boundary our model has learned and where the decision boundary is in reality.



The original decision boundary between the **red** and **blue** populations will misclassify the **orange** population as **blue** even though it should be **red** after the true boundary (the black line) shifts to the left.

How should model drift be addressed?

Now that you have gained familiarity with model drift and its common causes, the next step is learning how to detect it and address it. Proceed to the next video to learn more about model drift with a special focus on how to address it in the context of TinyML.