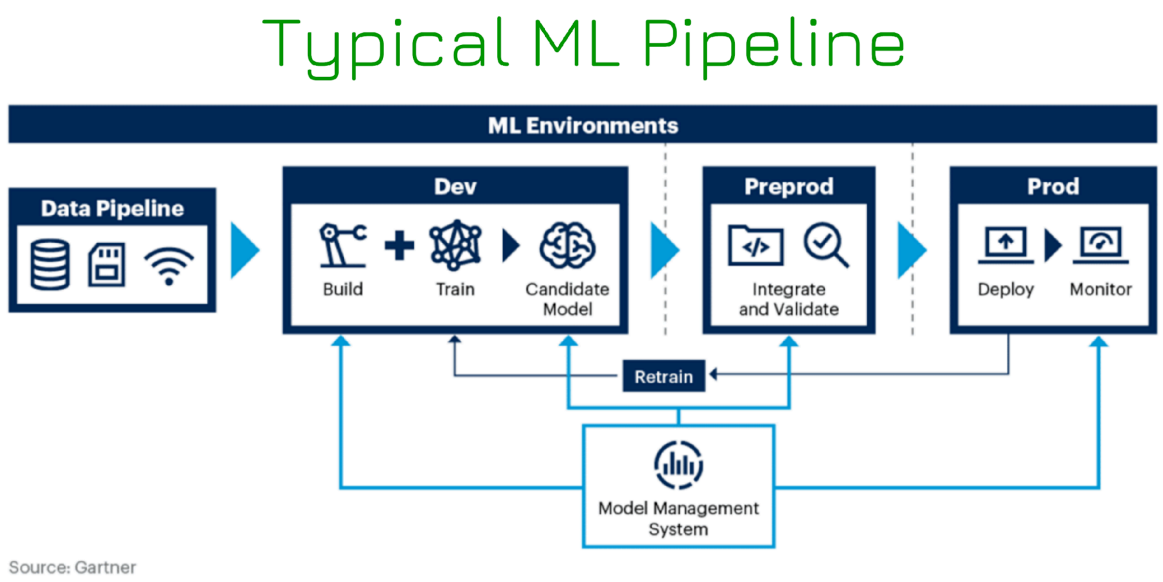
Digit Classification via Hand coded ML pipeline

Let's build a machine learning pipeline for a simple task of digit classification. Recall the typical blocks in the pipeline as discussed in the class.



The Dataset Pipeline

Consider downloading the digit classification dataset from any source of your convenience. There are multiple sources, one is Keras. The dataset contains 60000+10000=70000 images for handwritten digits.

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

Task: 1 (5 points)

Create a Python function, that rotates the 28x28 image by one of -30, -20, -10, +10, +20, +30 degrees. Essentially, the function should have a prototype as

```
def rotate(image, angle):
    # perform image rotation using a suitable method
    return rotated_image
```

Task: 2 (10 points)

Create a function that would generate an oversampled population of rotated images.

```
def generate_dataset(dataset, oversample_rate:float):
       oversample_rate = 2 if oversample_rate < 1 else oversample_rate
       to_generate_count = dataset.shape[0]*(oversample_rate-1.0)
       # now randomly pick a data point from the dataset
       # rotate the datapoint by any one of the angles randomly
       # add the new image to the dataset
       return updated_dataset
```

By now, you have a method to generate 3x times the original dataset size. The original data with 70k images, another 70k rotated from -30 degrees to +30 degrees. You should have 490k images now.

The Model Building Pipeline

Pick your favourite classification method to learn the classifier. Typically, a simple convolution network should do the trick for you. Consider the following implementation of the simple neural network classifier from https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/multi-class_classification_with_MNIST.ipynb?hl=en You can choose any method of your choice to learn the classifier.

Task: 3 (20 points)

Write a single wrapper function that should learn your classifier with hyperparameter tuning. Every model has hyper-parameters. For instance, a CNN model has the kernel size and number of layers as hyperparameters. For an ANN, the number of hidden nodes and layers are hyperparameters. Your function to test different hyperparameter settings to find the best configuration that maximizes the accuracy of classification.

```
def learn_model(dataset):
      # create the train-test cv splits and use the validation data for tuning.
      # find the best hyperparameter that maximized the perf
      return tuned_model
```

Monitoring the Deployment

Task: 4 (15 points)

Create a monitoring function that checks the performance of the model using a small sample of ground truth data. Essentially, hold out a small of data as the ground truth and never use that for training/validating the model. The hold out dataset is used to test the performance of the model after deployment. If the estimated performance on the sample is below a threshold, raise a flag to notify “drift”.

```
def monitor_perf(model, ground_truth_dataset, threshold):
      yhat = model.predict(ground_truth_dataset.x)
      # if the accumulated error E = |ground_truth_dataset.Y – yhat|>threshold
      # raise a flag!
      return drift_flag
```

End-to-End Evaluation

Chain the modules now in tandem.

1. For starters, use a portion of the original data to train the model and simulate the model deployment by running the monitoring script.
2. If the monitoring script reports error, you may have to retrain the model with more data until the monitoring script is happy.
3. Now, let's add a twist. Change the monitoring script to a sample that rotated -30°. Note that we are infusing a concept drift (changing the expectation of an end user).
4. Repeat the training process until the monitor (sensor) is happy.
5. Add likewise the -20°, -10°, +10°, +20°, +30° degree data into the mix.
6. At the end, your model will be matured enough to handle any orientation of the digit image, which is our end-goal.

Important Pointers

1. You should strictly follow the function prototypes when you build the functions.
2. You are expected to submit *.py files instead of Jupyter notebooks.
3. Here is the allocation of points per task
   10% for a clean coding style
   40% for the correctness of the implementation
   20% for readable comments
   10% for input arguments' validation/boundary check
   20% for unit test modules
4. You should also submit relevant performance reports (of your choice of measurements) with plots and their respective interpretations.

Best wishes.