The Dataset Pipeline

Task 1

```
from PIL import Image  # Import the Image class from the PIL library
import numpy as np  # Import NumPy for array manipulation

def rotate(image, angle):
    # Convert the input image to a NumPy array
    image_array = np.array(image)

    # Create a PIL Image object from the NumPy array
    pil_image = Image.fromarray(image_array)

    # Rotate the image using the provided angle and BICUBIC resampling
    rotated_image = pil_image.rotate(angle, resample=Image.BICUBIC)

    # Return the rotated image as a PIL Image object
    return rotated_image
```

Task 2

```
import random
import numpy as np

def generate_rotated_dataset(original_dataset, oversample_rate):
    """
    Generate an oversampled population of rotated images.

    Parameters:
    - original_dataset: Tuple (X, y) representing the original
dataset.
    - oversample_rate: Oversampling rate. If less than 1, set to 2.0.

    Returns:
    - updated_dataset: Tuple (X, y) representing the updated dataset.
    """

    X, y = original_dataset
    rotated_X = []  # List to store rotated images
    rotated_y = []  # List to store corresponding labels

    # Ensure oversample_rate is at least 2.0
    oversample_rate = max(oversample_rate, 2.0)

    # Calculate the number of images to generate for oversampling
    to_generate_count = int(X.shape[0] * (oversample_rate - 1.0))
```

```
    for _ in range(to_generate_count):
        # Randomly select a data point index from the original dataset
        idx = random.randint(0, X.shape[0] - 1)

        # Randomly choose a rotation angle from the set [-30, -20, -
10, 10, 20, 30]
        rotation_angle = random.choice([-30, -20, -10, 10, 20, 30])

        # Rotate the selected image using the provided rotate function
        rotated_image = rotate(X[idx], rotation_angle)

        # Convert the rotated image (PIL Image) to a NumPy array
        rotated_image = np.array(rotated_image)

        # Append the rotated image and its corresponding label to the
new dataset
        rotated_X.append(rotated_image)
        rotated_y.append(y[idx])

    # Convert the lists of rotated images and labels to NumPy arrays
    rotated_X = np.array(rotated_X)
    rotated_y = np.array(rotated_y)

    # Concatenate the original and rotated datasets along the first
axis
    updated_X = np.concatenate([X, rotated_X], axis=0)
    updated_y = np.concatenate([y, rotated_y], axis=0)

    return updated_X, updated_y
```

The Model Building Pipeline

Task 3

```
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV

def learn_model(dataset):
    """
    Wrapper function for model learning using XGBoost classifier with
hyperparameter tuning.

    Parameters:
    - dataset: Tuple (X, y) representing the dataset.

    Returns:
    - tuned_model: XGBoost classifier with the best hyperparameters.
```

```
    """

    # Unpack the dataset
    X, y = dataset

    # Reshape the input features if needed (XGBoost expects 2D array)
    X_train, _, y_train, _ = train_test_split(X.reshape(X.shape[0], -
1), y, test_size=0.2, random_state=42)

    # Define XGBoost Classifier
    xgb_classifier = XGBClassifier()

    # Define hyperparameters to tune
    param_grid = {
        'n_estimators': [100],
        'learning_rate': [0.01],
        'max_depth': [3],
        'subsample': [0.8, 1.0],
        'colsample_bytree': [0.8, 1.0],
    }

    # Use GridSearchCV for hyperparameter tuning
    grid_search = GridSearchCV(xgb_classifier, param_grid, cv=2,
verbose=3)
    grid_search.fit(X_train, y_train)

    # Return the tuned XGBoost classifier
    tuned_model = grid_search.best_estimator_
    return tuned_model
```

Task 4

```
from sklearn.metrics import accuracy_score

def monitor_performance(model, ground_truth_dataset,
accuracy_threshold):
    """
    Monitor the performance of a model based on accuracy and check for
concept drift.

    Parameters:
    - model: Trained machine learning model.
    - ground_truth_dataset: Tuple (X, y) representing the ground truth
dataset.
    - accuracy_threshold: Threshold for detecting concept drift.

    Returns:
    - drift_detected: Boolean indicating whether concept drift is
detected.
    """
```

```
    # Unpack the ground truth dataset
    X, y_true = ground_truth_dataset

    # Make predictions using the model on the ground truth dataset
    y_pred = model.predict(X.reshape(X.shape[0], -1))

    # Calculate the accuracy score
    accuracy = accuracy_score(y_true, y_pred)

    # Check if the accuracy drops below the threshold, indicating
potential concept drift
    drift_detected = accuracy < accuracy_threshold

    return drift_detected

# End-to-End Evaluation
from keras.datasets import mnist
import warnings
warnings.filterwarnings('ignore')
original_data, ground_truth_data = mnist.load_data()

# Task 2: Learn the model
best_model = learn_model(original_data)
```

```
Fitting 2 folds for each of 4 candidates, totalling 8 fits
[CV 1/2] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3,
n_estimators=100, subsample=0.8;, score=0.835 total time= 2.0min
[CV 2/2] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3,
n_estimators=100, subsample=0.8;, score=0.834 total time= 1.5min
[CV 1/2] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3,
n_estimators=100, subsample=1.0;, score=0.833 total time= 1.3min
[CV 2/2] END colsample_bytree=0.8, learning_rate=0.01, max_depth=3,
n_estimators=100, subsample=1.0;, score=0.831 total time= 1.3min
[CV 1/2] END colsample_bytree=1.0, learning_rate=0.01, max_depth=3,
n_estimators=100, subsample=0.8;, score=0.832 total time= 1.7min
[CV 2/2] END colsample_bytree=1.0, learning_rate=0.01, max_depth=3,
n_estimators=100, subsample=0.8;, score=0.831 total time= 1.8min
[CV 1/2] END colsample_bytree=1.0, learning_rate=0.01, max_depth=3,
n_estimators=100, subsample=1.0;, score=0.830 total time= 1.7min
[CV 2/2] END colsample_bytree=1.0, learning_rate=0.01, max_depth=3,
n_estimators=100, subsample=1.0;, score=0.830 total time= 1.8min
```

```
# Task 3: Monitor model performance

# Set the threshold for concept drift detection
threshold = 0.95

# Use the monitor_performance function to check for concept drift
drift_flag = monitor_performance(best_model, ground_truth_data,
threshold)
```

```python
# The variable 'drift_flag' now holds a boolean indicating whether
concept drift is detected.

# Optionally, we can take actions based on the drift detection result,
for example:
if drift_flag:
    print("Concept drift detected! Model performance may be
affected.")
    # Add actions to handle concept drift, such as retraining the
model, updating features, etc.
else:
    print("No concept drift detected. Model performance is stable.")
    # Continue with regular operations since no drift is detected.
```

```
Concept drift detected! Model performance may be affected.
```

```python
# Task 4: Handle drift and repeat training

# Unpack the original and ground truth datasets
train_imgs = (*original_data,)
grd_truth_imgs = (*ground_truth_data,)

# Iterate over rotation angles
for theta in [-30, -20, -10, 10, 20, 30]:
    rate = 2  # Set the initial oversample rate

    # Print information about the current rotation angle
    print(f'Adding new ground truth data with images rotated by
{theta}.')

    # Generate new rotated ground truth data
    grd_truth_imgs = generate_rotated_dataset(grd_truth_imgs,
oversample_rate=2)

    # Check for concept drift after adding new ground truth data
    drift_flag = monitor_performance(best_model, grd_truth_imgs,
threshold)

    # Print information about model performance after adding new
ground truth data
    print(f'Model performance is good after adding the new images. -
{drift_flag}.\n')

    # Continue training and monitoring for different oversample rates
    while not drift_flag:
        print('-' * 80)
        print(f'Augment train data with oversample rate = {rate}')

        # Generate new rotated training data
```

```
        train_imgs = generate_rotated_dataset(train_imgs,
oversample_rate=rate)

        # Learn a new model with the augmented training data
        best_model = learn_model(train_imgs)

        # Check for concept drift after training with augmented data
        drift_flag = monitor_performance(best_model, grd_truth_imgs,
threshold)

        # Print information about model performance after training
with the current oversample rate
        print(f'Model performs well for oversample rate {rate} and
rotation angle {theta}. - {drift_flag}.')
        print('-' * 80, '\n')

        # Increase the oversample rate for the next iteration
        rate += 1

# The code iterates over different rotation angles, adds new ground
truth data with rotated images and then continues training the model
with increasing oversample rates until concept drift is detected.
```

```
Adding new ground truth data with images rotated by -30.
Model performance is good after adding the new images. - True.

Adding new ground truth data with images rotated by -20.
Model performance is good after adding the new images. - True.

Adding new ground truth data with images rotated by -10.
Model performance is good after adding the new images. - True.

Adding new ground truth data with images rotated by 10.
Model performance is good after adding the new images. - True.

Adding new ground truth data with images rotated by 20.
Model performance is good after adding the new images. - True.

Adding new ground truth data with images rotated by 30.
Model performance is good after adding the new images. - True.
```