Practice activity: Logging

Disclaimer: *You will log errors and exceptions for debugging in this activity. If you are currently taking the course "Building Intelligent Troubleshooting Agents," proceed with this activity.*

If you are taking the course "Microsoft Azure for AI and Machine Learning" and already took the course "Building Intelligent Troubleshooting Agents," you can skip this activity and use that file. Otherwise, you must complete this activity in order to move forward.

Introduction

Imagine you're debugging a complex machine learning model, and a critical error occurs during model training or prediction. Without logging, tracking down what went wrong can feel like finding a needle in a haystack. That's where logging becomes invaluable. Logging not only allows you to monitor your system's behavior but also helps you capture key events and errors for future analysis and debugging. This is an essential tool to ensure the smooth operation of any machine learning system.

By the end of this activity, you will be able to:

1. Set up Python's logging module to capture and store logs.
2. Implement logging at key stages of the machine learning pipeline, including data preprocessing, model training, and predictions.
3. Log errors and exceptions to a file for debugging purposes.

Step-by-step process to implement logging in machine learning systems

Create a new Jupyter notebook. Make sure you have the appropriate Python 3.8 Azure ML kernel selected.

The remaining of this reading will guide you through the following steps:

- Step 1: Set up logging
- Step 2: Log data preprocessing
- Step 3: Log model training
- Step 4: Log predictions and inference
- Step 5: Log errors and exceptions

Step 1: Set up logging

First, configure the logging module to capture logs and store them in a file. You will log important events such as when the system starts, when data preprocessing occurs, and when model training is completed.

Instructions

1. Import the logging module.
2. Set up the logging configuration to store logs in a file with the desired log level, in this case the INFO level.

Code example

```
logging.info("Logging setup complete.")
```

Explanation

Setting up logging at the start of the program facilitates the capturing of important events, warnings, and errors throughout the machine learning pipeline. Setting the level of logging to “INFO” ensures that all logs of priority level INFO or higher will be recorded—logs at the DEBUG level or lower will not be recorded.

Your log file, ml_pipeline.log, will contain messages logged using the logging.info() or logging.error() functions. You can access it by clicking on the file in the Notebooks view in the Azure ML Studio; you may need to click the “refresh” button in the file viewer panel before you can see the newly created log file.

Step 2: Log data preprocessing

During data preprocessing, log key steps such as loading the dataset, handling missing values, and normalizing data. Logging these activities helps to track how the data is being prepared for the machine-learning model.

Instructions

1. Log the start and completion of data loading and preprocessing.
2. Capture any transformations applied to the data, such as filling missing values or scaling.

Code example

```
import pandas as pd

# Log the start of data loading
logging.info("Loading dataset...")

# Load the dataset
df = pd.read_csv('your-dataset.csv')
logging.info("Dataset loaded successfully.")

# Log the start of preprocessing
logging.info("Starting data preprocessing...")

# Example preprocessing: handling missing values
df.fillna(0, inplace=True)
logging.info("Missing values filled with 0.")

# Log the completion of preprocessing
logging.info("Data preprocessing completed.")
```

Explanation

Logging during data preprocessing ensures that you have a detailed record of the steps taken to clean and prepare the data. This is useful for tracking any changes made to the dataset. Each time you call logging.info(), a new log is written to the log file you specified earlier.

Step 3: Log model training

During model training, log the start and end of the training process, as well as key metrics such as training accuracy or loss. If an error occurs during training, log the exception and capture relevant details.

Instructions

1. Log the start of model training.
2. Capture any key metrics, such as training accuracy or loss.
3. Log the completion of training or any errors that occur.

Code example

```
from sklearn.tree import DecisionTreeClassifier

# Log the start of model training
logging.info("Starting model training...")

try:
    # Train the decision tree model
    model = DecisionTreeClassifier()
    model.fit(X_train, y_train)
```

```
        model.fit(X_train, y_train)
        logging.info("Model trained successfully.")
    except Exception as e:
        logging.error(f"Error during model training: {e}")

    # Example logging of training accuracy (if applicable)
    accuracy = model.score(X_train, y_train)
    logging.info(f"Training accuracy: {accuracy:.2f}")
```

Explanation

By logging the training process, you can monitor the system's progress and capture critical metrics, such as accuracy. Any errors that occur during training are also logged for debugging. Because our logging object's logging level was configured as INFO, and the level ERROR is above that of the level LOG, calling logging.error() will write an error message to our log file.

Step 4: Log predictions and inference

When making predictions or performing inferences, log the inputs and outputs to track the system's behavior. If an error occurs during prediction, log it and continue.

Instructions

1. Log the start of the prediction process.
2. Capture the input features and prediction outputs.
3. Log any errors that occur during inference.

Code example

```
# Log the start of predictions
logging.info("Starting model predictions...")

try:
    # Make predictions
    predictions = model.predict(X_test)
    logging.info("Predictions made successfully.")
except Exception as e:
    logging.error(f"Error during predictions: {e}")

# Log the output (in production systems, limit the amount of data logged)
logging.info(f"Prediction output: {predictions[:5]}")  # Log only first 5 predictions
```

Explanation

Logging predictions helps to trace the behavior of the model during inference. By capturing inputs and outputs, you can ensure that the model is functioning as expected and investigate any errors that occur.

Step 5: Log errors and exceptions

Ensure that any errors or exceptions are logged in detail. This includes logging the type of error, the message, and any relevant context for debugging.

Instructions

1. Use try-except blocks to catch errors.
2. Log the error message and any additional information needed for debugging.

Code example

```
# Example: logging an exception during data validation
def validate_data(data):
    try:
        if not isinstance(data, pd.DataFrame):
            raise ValueError("Input must be a pandas DataFrame.")
        logging.info("Data validation successful.")
    except ValueError as e:
        logging.error(f"Data validation error: {e}")

# Validate the dataset
validate_data(df)
```

Explanation

Capturing detailed logs of errors and exceptions helps developers identify the root cause of issues quickly, making debugging more efficient.

Conclusion

Logging is a crucial component of maintaining and debugging machine learning systems. It allows you to track system behavior, monitor key events such as data preprocessing and model training, and log errors or exceptions for future debugging. By implementing comprehensive logging at every stage of the pipeline, you'll be equipped to quickly diagnose issues, track progress, and maintain system stability as you work with complex machine-learning models.