**Detailed Documentation**

**Occupational Code Classification MLOps Pipeline**

**Table of Contents**

1. [Overview](#overview)
2. [Pipeline Flow](#pipeline-flow)
3. [Element Breakdown](#element-breakdown)
   * [Directories and Their Roles](#directories-and-their-roles)
   * [Key Scripts and Modules](#key-scripts-and-modules)
4. [Integration and Workflow](#integration-and-workflow)
   * [Data Ingestion to Preprocessing](#data-ingestion-to-preprocessing)
   * [Model Training and Evaluation](#model-training-and-evaluation)
   * [Deployment and Inference](#deployment-and-inference)
   * [Tracking and Versioning](#tracking-and-versioning)
5. [Conclusion](#conclusion)

**Overview**

The **Occupational Code Classification MLOps Pipeline** is designed to automate the process of building, training, evaluating, and deploying machine learning models for classifying 4-digit occupational codes. The pipeline is modular, allowing for independent development and testing of each component, and is built with scalability and maintainability in mind.

**Pipeline Flow**

1. **Data Ingestion**: Load raw data from CSV files or a MSSQL database.
2. **Data Validation**: Validate the integrity and schema of the ingested data.
3. **Data Preprocessing**: Clean the data and perform feature engineering.
4. **Model Training**: Train machine learning models using the preprocessed data.
5. **Model Evaluation**: Evaluate model performance using appropriate metrics.
6. **Pre-Production Testing**: Conduct final tests before deploying to production.
7. **Deployment**: Deploy the model to a production environment.
8. **Inference**: Use the deployed model to make predictions on new data.
9. **Monitoring and Tracking**: Monitor model performance and track versions.

**Element Breakdown**

**Directories and Their Roles**

**1. azure\_pipelines/**

* **Purpose**: Contains configuration files for setting up CI/CD pipelines using Azure DevOps.
* **Functionality**: Automates testing, building, and deployment processes, ensuring consistent and reliable integration of new code.

**2. config/**

* **Purpose**: Stores configuration files and scripts.
* **Files**:
  + **config.yaml**: Centralized settings for the project, such as data paths, model parameters, and environment variables.
  + **configScript.py**: A script to load and manage configurations throughout the project.
  + **logging.yaml**: Configures logging formats and levels for consistent logging across modules.

**3. data/**

* **Purpose**: Houses datasets at different processing stages.
* **Subdirectories**:
  + **raw/**: Contains unprocessed, original data.
  + **interim/**: Stores data that is in the midst of processing steps.
  + **processed/**: Contains data ready for modeling after preprocessing.
  + **sourceModels/**: Holds pre-trained models or embeddings required for advanced modeling techniques.

**4. mlflow/**

* **Purpose**: Integrates MLflow for experiment tracking and model management.
* **Files**:
  + **mlproject.yaml**: Defines MLflow project specifications, entry points, and dependencies.

**5. notebooks/**

* **Purpose**: Contains Jupyter notebooks for data exploration and prototyping.
* **Files**:
  + **test.ipynb**: An example notebook demonstrating EDA or testing code snippets.

**6. scripts/**

* **Purpose**: Contains executable scripts that orchestrate pipeline stages.
* **Subdirectories**:
  + **main\_pipeline/**: Houses the main script that runs the entire pipeline sequentially.
  + **main\_scripts/**: Contains scripts for individual stages like preprocessing, training, and evaluation.
  + **shell\_scripts/**: Shell scripts to execute the Python scripts, facilitating automation and scheduling.

**7. src/**

* **Purpose**: Core source code directory, containing all the modules and packages.
* **Subdirectories**:
  + **data\_ingestion/**:
    - **data\_loader.py**: Module for loading data from files or databases.
    - **data\_validator.py**: Ensures data integrity and correct schema.
  + **data\_preprocessing/**:
    - **data\_cleaner.py**: Cleans data by handling missing values, duplicates, and outliers.
    - **feature\_engineer.py**: Creates new features or transforms existing ones to enhance model performance.
  + **models/**:
    - **bert\_model.py**: Implements the BERT model for handling textual data.
    - **xgb\_model.py**: Implements the XGBoost model, suitable for tabular data.
  + **training/**:
    - **trainer.py**: Coordinates the training process, including model selection and hyperparameter tuning.
  + **evaluation/**:
    - **evaluator.py**: Assesses model performance using metrics like accuracy, precision, and recall.
  + **inference/**:
    - **predictor.py**: Uses the trained model to make predictions on new, unseen data.
  + **pipeline/**:
    - *(Can include orchestrators and utilities for managing data flow between stages.)*
  + **utils/**:
    - *(Contains helper functions, such as logging setups, configuration loaders, and common utilities.)*

**8. tests/**

* **Purpose**: Contains all test scripts ensuring code functionality and reliability.
* **Subdirectories**:
  + **unit/**: Tests individual components or functions in isolation.
  + **integration/**: Tests the interaction between different components to ensure they work together correctly.
  + **qa/**: Focuses on quality assurance, including performance testing and data validation.

**9. tracking/**

* **Purpose**: Manages versioning and tracking of data and models.
* **Subdirectories**:
  + **data\_versioning/**: Logs changes to datasets, ensuring reproducibility.
  + **model\_versioning/**: Keeps track of different model versions and their performance metrics.
  + **production\_tests/**: Stores results from tests conducted in the production environment.

**10. training\_logs/**

* **Purpose**: Stores logs generated during the training process, useful for debugging and monitoring training progress.

**Key Scripts and Modules**

**config/config.yaml**

* **Purpose**: Acts as the single source of truth for configuration settings.
* **Functionality**: Stores settings like file paths, database connections, model hyperparameters, and environment variables.

**src/data\_ingestion/data\_loader.py**

* **Purpose**: Loads data from various sources.
* **Functionality**: Contains methods to read data from CSV files or query databases, returning data in a structured format for further processing.

**src/data\_ingestion/data\_validator.py**

* **Purpose**: Validates the integrity and structure of the ingested data.
* **Functionality**: Checks for missing values, data type mismatches, and schema compliance, ensuring that downstream processes receive clean data.

**src/data\_preprocessing/data\_cleaner.py**

* **Purpose**: Cleans the raw data.
* **Functionality**: Handles missing values, duplicates, and outliers, and may perform initial data transformations.

**src/data\_preprocessing/feature\_engineer.py**

* **Purpose**: Enhances data with new features.
* **Functionality**: Creates new variables or transforms existing ones to improve model performance, such as encoding categorical variables or creating interaction terms.

**src/models/bert\_model.py**

* **Purpose**: Implements the BERT model for text classification tasks.
* **Functionality**: Defines the architecture and methods for training and inference using BERT, handling tokenization and embedding of textual data.

**src/models/xgb\_model.py**

* **Purpose**: Implements the XGBoost model for classification tasks.
* **Functionality**: Defines the model architecture, hyperparameters, and methods for training and prediction using XGBoost.

**src/training/trainer.py**

* **Purpose**: Orchestrates the model training process.
* **Functionality**: Coordinates data preparation, model instantiation, training loops, and saving trained models.

**src/evaluation/evaluator.py**

* **Purpose**: Evaluates trained models.
* **Functionality**: Uses test datasets to compute performance metrics and generates evaluation reports.

**src/inference/predictor.py**

* **Purpose**: Performs model inference on new data.
* **Functionality**: Loads trained models and applies them to incoming data to generate predictions.

**scripts/main\_pipeline/main\_pipeline.py**

* **Purpose**: Serves as the entry point for running the entire pipeline.
* **Functionality**: Sequences the execution of all pipeline stages in the correct order.

**Integration and Workflow**

**Data Ingestion to Preprocessing**

1. **Data Loading**
   * **Process**: The data\_loader.py module reads data from specified sources based on configurations.
   * **Integration**: Utilizes parameters from config.yaml to locate data files or database credentials.
2. **Data Validation**
   * **Process**: The data\_validator.py module checks the loaded data for compliance with expected formats.
   * **Integration**: If data passes validation, it moves to the interim data storage; otherwise, errors are logged.
3. **Data Cleaning**
   * **Process**: The data\_cleaner.py module cleans the data, handling anomalies.
   * **Integration**: Cleaned data is saved in the interim/ directory for further processing.
4. **Feature Engineering**
   * **Process**: The feature\_engineer.py module creates or transforms features.
   * **Integration**: Outputs processed data ready for modeling, stored in the processed/ directory.

**Model Training and Evaluation**

1. **Model Selection**
   * **Process**: Depending on the task, either the BERT model (bert\_model.py) or the XGBoost model (xgb\_model.py) is selected.
   * **Integration**: Model configurations are read from config.yaml, allowing for dynamic model selection.
2. **Training**
   * **Process**: The trainer.py module handles the training process, including data splitting, model instantiation, and training loops.
   * **Integration**: Training parameters and hyperparameters are sourced from configurations.
3. **Evaluation**
   * **Process**: The evaluator.py module assesses model performance using test datasets.
   * **Integration**: Generates reports and logs performance metrics to training\_logs/.

**Deployment and Inference**

1. **Pre-Production Testing**
   * **Process**: Before deployment, models undergo rigorous testing stored in tests/qa/ and tracking/production\_tests/.
   * **Integration**: Ensures models meet performance thresholds and are stable.
2. **Deployment**
   * **Process**: The production.sh script automates the deployment of the model to a production environment.
   * **Integration**: Deployed models are versioned and tracked via model\_versioning/.
3. **Inference**
   * **Process**: The predictor.py module uses the deployed model to make predictions on new data.
   * **Integration**: Supports real-time or batch inference, integrating with data ingestion modules if needed.

**Tracking and Versioning**

* **Data Versioning**
  + **Process**: Any changes to datasets are logged in data\_versioning/.
  + **Integration**: Allows for reproducibility and auditing of data changes over time.
* **Model Versioning**
  + **Process**: Each trained model version is logged with metadata about training conditions and performance.
  + **Integration**: Facilitates rollback to previous models if necessary and comparison between model versions.
* **Experiment Tracking**
  + **Process**: MLflow or similar tools are used to track experiments, parameters, and metrics.
  + **Integration**: Ensures transparency and ease of experimentation.

**Conclusion**

This detailed documentation provides a comprehensive understanding of the **Occupational Code Classification MLOps Pipeline**. By breaking down each component and explaining its role within the pipeline, we've outlined how data flows from raw ingestion to model deployment. The modular design supports scalability and maintainability, allowing for individual components to be developed, tested, and deployed independently while ensuring they integrate seamlessly into the overall workflow.

By adhering to best practices in MLOps, this pipeline not only facilitates efficient model development but also ensures robustness, reproducibility, and compliance with data governance standards.