

APS SENSOR FAULT DETECTION

PROJECT

1. Setup.py

↳ basic structure and metadata for the project Package to be used for distribution and installation facilities.

↳ setup func : → distribution metadata
→ package details
↳ dependencies
↳ versions
↳ author info etc.

2. Requirements

↳ requirements.txt
↳ lib == version
↳ -e : find packages

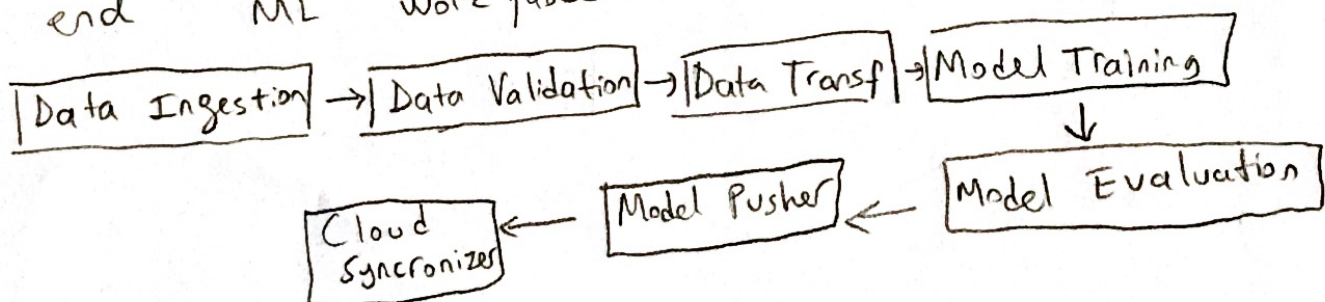
———— So far, its basic package structure ————

3. Create source folder ~~sensor~~ and turn it into a package

↳ Create sensor package

↳ Create pipeline package in sensor folder

↳ high level, series of interconnected data processing and modeling steps designed to streamline the end-to-end ML work flow



4. Create components package under sensor folder:

- ↳ data-ingestion.py
- ↳ data-validation.py
- ↳ data-transformation.py
- ↳ model-evaluation.py
- ↳ model-pusher.py
- ↳ model-trainer.py

5. Create data-access package under sensor folder:

↳ sensor-data.py

↳ SensorData class:

↳ init : mongo-client DB connection

↳ save-csv-file : read data from filepath then save it to a Database collection

↳ export-collection-as-dataframe : read data from db collection and return as df.

6. Create configuration package under sensor folder

↳ configuration package is often used to organize and manage various configuration settings and parameters for projects. In this project we created this package only to handle MongoDB connections centrally.

↳ mongo-db-connection.py

↳ class MongoClient

7. Create constants package under sensor folder.

- ↳ centralize and organize constant values, config settings or other static data that are used across different parts of our project.
- ↳ improves code readability, maintainability, and consistency by providing a single place to define and manage these values.
- ↳ application.py : APP_HOST , APP_PORT
- ↳ database.py : DATABASE_NAME, COLLECTION_NAME
- ↳ env-variables.py : MONGODB_URL_KEY, AWS ID KEY & SECRET KEY
(sensitive variables)
- ↳ s3-bucket.py : AWS-REGION
TRAINING-BUCKET-NAME
PREDICTION-BUCKET-NAME

8. Create exceptions.py file in sensor folder:

- ↳ def error-message-detail
- ↳ class SensorException (Exception)
 - ↳ our customized exception class
 - ↳ structured code
 - ↳ Consistency across the project
 - ↳ Encapsulation

9. Create `logger.py` file in `sensor` folder

- ↳ logs-path → make dirs $\xrightarrow{\text{Join}}$ LOG-FILE-PATH
 LOG-FILE
 (name w/ timestamp)

- ↳ logging.basicConfig (filename, format, level)

→ Capture and store information to aid in monitoring, debugging, trouble shooting, and auditing applications.

10. Create entity package under sensor package *

↳ artifact - entity . py

- ↳ datatypes serving as clear and structured representations of the different artifacts (outputs) that our app deals with during various stages of the ML cycle / pipeline.
- ↳ By encapsulating the data and behaviour within these classes, we create more organized and understandable way to manage the complexity of our ML pipeline
- ↳ Each data class defines attributes that correspond to specific properties or metadata related to the artifacts, and the @dataclass decorator simplifies the creation of these classes by automatically generating common methods like `--init--` and `--repr--`
- ↳ Promotes : Readability, Maintainability and separation of concerns (Abstraction), (Encapsulation)
- ↳ Check the Artifact Diagram

~~11.~~ ↳ config - entity . py

- ↳ define set of Configuration classes, each encapsulating different configuration settings related to various stages of our training pipeline. These classes help us to organize and manage the configuration settings required for different parts of our application.
- ↳ contains sets of directory paths, file paths, object paths and some constants (metrics, threshold values etc) derived from constants > training - pipeline

11. Create ml package under sensor folder

- * ↳ Custom ml algorithms
- * ↳ Custom loss functions
- * ↳ Custom metric functions
- * ↳ Model evaluation functions
- * ↳ visualizations
- * ↳ Documentation & Research: We can store here our research documents, experimental setups and findings.
- * ↳ Experiment Tracking
- * ↳ Model Version Control

We've used ml package for 2 specific reasons:

- ↳ classification Metrics
- ↳ model-estimator

↳ ml > metric > classification-metric.py

↳ Return Classification Metric Artifact:

$\left[\begin{array}{l} * f_1\text{-score} \\ * \text{precision-score} \\ * \text{recall-score} \\ \vdots \end{array} \right]$

↳ ml > model > model-estimator.py

↳ Target Value Mapping: encoding / decoding the target feature for prediction & evaluation : **OPTIONAL**

↳ class SensorModel: ^{preprocessing pipeline}
def init: - processor and model

def predict: transform by preprocessor → predict
and return \hat{y}

↳ class Model Resolver:

↳ look into the Saved models directory

↳ def is_model_exists → check if there is a saved model dir & a model in it

↳ def get_best_model_path: then get best-model-path
(logic: latest Dno.)