# Nitrogen Fertilization model training infrastructure user guide

Written by Itay Lotan at May 2024

**Table of contents:**



## Introduction

This document discusses the training infrastructure of the Nitrogen Fertilization model. This model is a regression model that predicts the amount of nitrogen expected to be found in a farming sector, based on data points like the properties of its ground and historical precipitation data of its surrounding area. The role of the model training infrastructure is to provide a simple and robust abstraction layer for:

- The preprocessing of the dataset
- The training and evaluation of the the model
    - the setup of the model training and evaluation can be configured easily by adjusting the input of the infrastructure, which is described below
- The ability to inspect and analyze the model training process
    - one the model training process outputs is a highly detailed training report, which is an HTML file that's divided into multiple sections and contains textual information as well as various useful plots and attachments
- The storage of the trained model weights and metadata as an artifact file
- The usage of the model artifact file for the inference phase (to make new predictions)

## Installation

### Prerequisites

- Running from one of the supported operating systems, which are
    - Mac OS (Intel chip or Apple Silicon chip, both are supported)

    - Linux Debian 11 (tested over the docker image python:3.10.14-bullseye)
    - Other operating systems may be supported as well, but the infrastructure wasn't tested in them
- Python version 3.10 installed
    - Including pip version 19 or above, which should be installed by default with Python 3.10
- Optional - an isolated python virtual environment for the installation of the infrastructure
    - This approach is recommend in order to avoid the collision with conflicting 3rd party dependencies of other installed packages
    - For more details, see the official documentation, or use another similar tool like Poetry

Installation instructions

The infrastructure is built as a strand python package that's called 'bi-nitrogen-fertilization-ml-pipeline'. The specifications of the package are declared at the pyproject.toml file of the repository. This means that the infrastructure can be installed through pip or any other python packages installer (like Poetry).

For the purposes of this project, there are two main approaches for working with the infrastructure package - (1) install it like any other package, or (2) build the user scripts (or Jupyter notebooks) directly within the git repository of the package. Option 1 is the more standard approach, with its main downside being the more complex process of making changes in infrastructure's code. For every such change, a new version of the package will need to be released (i.e. committed and pushed to the infrastructure repo), and then the infrastructure package will need to be reinstalled by the user. Option 2 allows the developer to make changes in the infrastructure's code that will take effect immediately. Its main downside being the risk of coupling of the infrastructure code and the user experiments code.

Instructions for option 1 - install the infrastructure package as a standard python package

```
pip install --upgrade --force-reinstall
"git+https://github.com/itayL1/bi-nitrogen-fertilization-ml-pipeline.git@main"
```

This command installs the latest version of the infrastructure package directly from github (from the branch 'main'). If any version of the package is already installed, this command will first uninstall it, and then install the latest version. As discussed above, this command will

need to be reused for the retrieval of every desirable change that was made in the infrastructure's code.

Instructions for option 2 - develop the user code within the infrastructure git repository

1) Clone the git repository of the infrastructure:

```
git clone https://github.com/itayL1/bi-nitrogen-fertilization-ml-pipeline.git
```

2) Navigate to the repository's folder in the terminal

3) Install the infrastructure package in 'editable' mode using this command

```
pip install -e .
```

This command installs all the third party dependencies of the package, but keeps the actual code of the git repository as the installed package (in this case, for the package 'bi_nitrogen_fertilization_ml_pipeline'). For more details about installation of packages in 'editable' mode, see the relevant docs.

3.1) The command in step 3 will need to be reused only when the third party dependencies of the package are changed (i.e. when a new dependency is added, or a version of one of the dependencies is changed).

* Note that the infrastructure github repository is public, so no authentication is required For the installation commands.

**The Model Training component**

Python API

The main function that is exposed by the training component is called *train_and_evaluate_model*. This function can be imported directly from the package's main API:

```
from bi_nitrogen_fertilization_ml_pipeline.main_api import train_and_evaluate_model
```

This function receives the following arguments:

- *raw_train_dataset_df* - a pandas DataFrame that contains the entire train dataset, including the target column
- *features_config_dict* - a nested dictionary that contains the configuration of the features and target of the trained model. See the features config API reference for explanation about the schema of this argument.

- train_params_dict - a nested dictionary that defines the settings of the training process. See the train params API reference for explanation about the schema of this argument.
- *output_model_file_path* - the full path of the output trained model asset file. If the training process has finished successfully, the model file will be created in this path. The model file name must have the .zip extension.

Output 1 - the model file

The model file is a compressed folder that contains all the necessary assets for loading the model trained model and applying the preprocessing phase on an input dataset before invoking it. These assets contain:

- The trained model architecture
- The trained model weights
- The train dataset preprocessing artifacts

This model file can be passed later on to the inference API to make new predictions.

Output 2 - the pipeline report

At the same folder of the output model file, the final train pipeline report is created as well. This report is a stand-alone HTML file that is generated using the DataPane framework. In addition to textual information about the pipeline execution and results, the report also includes various graphs and attachments.

The report is divided into these sections:

- Summary - contains general information about the metadata and results of the training process
- Dataset EDA - contains the input dataset EDA reports as attachments. These reports are generated using the YData Profiling framework
- Dataset Preprocessing - contains information about the dataset preprocessing phase, like the imputation funnel and the statistics of the encoding of the categorical features
- Model Evaluation - contains the results of the model evaluation process, in detail and summarized using aggregations and graphs
- Final Model - contains information about the final model, its architecture, training graphs and feature importance. The feature importance is extracted using Shap.
- Warnings - contains a table of issues that occurred during the execution of the pipeline, but weren't critical enough to fail it

To see an example for a generated report, download this report file from the API reference and open it in a browser.

Usage example

See this notebook that contains an example for the usage of the training component.

## The Inference component

Python API

The main function that is exposed by the training component is called *predict_using_trained_model*. This function can be imported directly from the package's main API:

```
from bi_nitrogen_fertilization_ml_pipeline.main_api import predict_using_trained_model
```

This function receives the following arguments:

- *raw_inference_dataset_df* - a pandas DataFrame that contains the entire inference dataset, without the target column.
- *stored_model_file_path* - the path of a model file that was created by the model training component. This model will be used to make the predictions on the inference dataset.

Output

The return value of this function is a pandas Series that contains the model predictions for the inference dataset, in the same order (and with the same index) of the input inference dataset DataFrame.

Usage example

See this notebook that contains an example for the usage of the inference component.

## The assets module

The role assets module is to contain artifacts that are created during the development of this project and are worth preserving. Currently, it contains only the baseline model. The baseline model can be imported this way:

```
from bi_nitrogen_fertilization_ml_pipeline.assets import init_baseline_model
```

## An overview of infrastructure's core modules

The codebase of the infrastructure is divided into 5 submodules:

- The *core* module - which contains modules and data classes that are shared over the entire codebase. For example, the *model_storage* and *dataset_preprocessing* modules, and the *FeaturesConfig* and *TrainParams* data classes.
- The *model_training* module - which contains the implementation of the model training and evaluation flows, as well as train pipeline report creation.
- The *model_inference* module - which contains the implementation of the model inference API that was described above. This module is relatively simple, because it mostly uses high level utilities that were already implemented in the *core* module, besides making some basic validations on its input.
- The *main_api* module - which serves as the index module of the package, and exposes the final API functions that are intended to be used by the user (i.e. *predict_using_trained_model* and *predict_using_trained_model*).
- The *assets* module - which currently contains only the implementation of the baseline model.

Another thing to note - this package utilizes Pydantic for defining the scheme of its main API. when an input with invalid scheme is passed to it (mainly in the features_config or train_params arguments), in most cases an API error will be raised with a detailed Pydantic input validation error as its internal error.