University Master's Degree in
Machine Learning ML

Final project: Fully automated ML project

The main objective of the final project is to develop a fully automated machine learning project. This project will be developed in groups of three persons (except one that will be composed of two persons). Each group will have to

- Select an imbalanced binary classification problem (tabular data)
    - Be careful with the number of input features because you will have to develop the Haggin Face space as well.
- Select a proper model validation scheme
    - Model validation methodology.
    - Proper metrics: 1) threshold independent to optimize the values of the hyper-parameters and 2) threshold dependent to select the threshold to be used in production.
- Train the XGBoost model (its paper is attached to MiAulario's folder) and optimize the values of its hyper-parameters using Optuna.
- Accomplish the experiment tracking and model versioning using MLFlow
    - Log each trial (run) related to the different combinations of hyper-parameters tested with Optuna
        - You must think of the different metrics and artifacts you will log.
        - You can find information on the integration of Optuna and MLFlow in these sites
            - https://mlflow.org/docs/latest/ml/traditional-ml/tutorials/hyperparameter-tuning/
            - https://mlflow.org/docs/latest/ml/traditional-ml/tutorials/hyperparameter-tuning/notebooks/hyperparameter-tuning-with-child-runs/
    - Log the artifact(s) related to the global interpretability of the model.
- Serialize the best method using the proper format (for this method) for serialization.
- Develop the inference of the model as well as the API to serve it in production
    - You do not need to develop a CLI.
- Program the GUI and upload it to Hugging Face spaces.
- Test the components of the model.
- Develop the CI/CD pipeline using GitHub actions (as in the previous labs) to test the project, containerize the project, register the image, launch the web service, etc...
    - You must establish stages to train and serialize the model. To do it, in both steps, it is important to establish the tracking URI so that it is gotten from the environment
        - In the training and serialization scripts you should configure the tracking URI as

        - *mlflow.set_tracking_uri(os.environ.get("MLFLOW_TRACKING_URI", "mlruns"))*
    - In the .yml file you must include
        - In the environment variables of the job
            - *MLFLOW_DIR: ${{ github.workspace }}/mlruns*
        - In the environment (*env*) of the step
            - *MLFLOW_TRACKING_URI: ${{ env.MLFLOW_DIR }}*
    - Add a final step in the job to remove the MLFlow directory (*MLFLOW_DIR*)
- Monitor the model performance using Prometheus, to scrape the production metrics, and Grafana, to develop dashboards to visualize the previous scraped metrics. To this end, you must instrumentalize the API so that a new *endpoint* named ***metrics*** is exposed
    - Use the *prometheus_client* library.
    - Add *Counters* of *Gauges* to monitor different metrics.
    - The *metrics endpoint* only needs to return *generate_latest() (*method from *prometheus_client)*.

These are the minimum requisites to pass the final project. Besides them, you can do whatever you consider, for instance

- In the monitoring of the method, you can also
    - Detect data and concept drift
        - You can simulate them (random numbers).
        - You can simulate them using real or synthetic data. In this scenario, you can develop methods to compute the "real" data and concept drift.
        - In both cases, you can program a function to simulate it and call the function in the *metrics endpoint*. *Prometheus* scrapes the *endpoint* periodically and each time, it executes it. Therefore, the simulation would be run each time the *endpoint* is scrapped.
    - Detect fairness issues
    - Complete the automatization of the ML project performing automatic retraining when detecting data or concept drift (you can use *alertmanager* to this end).
- Consider the TabNet classifier (its paper is also attached to MiAulario's folder).
- Calibrate the model(s)
    - You can use the CalibratedClassifierCV from Scikit-learn.
- Orchestrate the project using Airflow.

Deliverables:

- A **report** with, at least,
    - a description of the classification problem to be tackled;

    - the justification of the model validation scheme used;
    - the logic of the testing;
    - the decision made to accomplish the task;
    - an analysis of the performance of the method(s);
    - screenshots of the monitorization of the method;
    - links to the GitHub repository and to the Hugging Face space.
- A **presentation**, which will be orally exposed the last session of the subject (9th of January). Maximum of 12 minutes per group.

Notes about the installation of Prometheus if you find problems with its installation from Docker. You can follow the following steps

- Download the compressed package
  - *wget https://github.com/prometheus/prometheus/releases/download/v2.50.1/prometheus-2.50.1.linux-amd64.tar.gz*
- Extract its content
  - *tar xvfz prometheus-2.50.1.linux-amd64.tar.gz*
- In the terminal, go to the uncompressed folder
- Configure *Promehetus* by modifying the p*rometheus.yml* file
  - Establish the scraping interval
  - Add a job in the scrape_configs section so that you specify the *URL* of the *API* as the target and you establish the *metrics* (*/metrics*) *endpoint* as *metrics_path*
- Execute the binary
  - *./prometheus --config.file=prometheus.yml*
- Check the scrapring
  - Open *http://localhost:9090* in a web browser.
  - In the menu, go to Status and then to Targets. Look for the name of the job you have configured in the .yml file and ensure its state is UP.
  - Query the metrics
    - In the menu, go to Graph
    - Write the name of the metrics you want to track (those defined in the Counters or Gauges and the API) and execute them.

Notes about the installation of Grafana if you find problems with its installation from Docker. You can follow the following steps

- Download the compressed package
  - *wget https://dl.grafana.com/oss/release/grafana-10.4.3.linux-amd64.tar.gz*
- Extract its content
  - *tar -zxvf grafana-10.4.3.linux-amd64.tar.gz*
- In the terminal, go to the uncompressed folder
- Execute the binary
  - *./grafana-server*
- Connect Grafana to Prometheus
  - Open *http://localhost:3000* in a web browser.
  - Both the username and the default password are *admin*.
  - Add a data source (left panel)
    - Select Prometheus from the list
    - Assign it a name and introduce the URL (http://localhost:9090)
    - Click the “save and test” button
  - Configure the dashboards to visualize the metrics
    - In the left panel go to Dashboards and click on “+ New Dashboard”

- Click on “+ Add visualization”
    - Ensure your Prometheus data source is selected
    - Write the name of the metric you want to visualize in the dashboard