Project 2: Big Data Processing (Predictive analytics)

Brief Summary

The Data Warehouse created in the first project is meant to support business intelligence (descriptive analysis via OLAP), reporting, and data analysis. As a continuation, in this project, you are tasked with developing an analytical pipeline for predicting **if an aircraft is going to interrupt its operation unexpectedly in the next seven days (i.e., a predictive analysis)**. In the AMOS database, there exists a table called Operation Interruption, which reports about past events. The aim of this project is to predict (and avoid happening) potential future occurrences in this table.

To this end, we will continue working with the ACME flying use case and the sources we are going to consider are the following:

1. The data warehouse developed for the first lab, which provides a consolidated view of the AMOS and AIMS databases.
2. Additional CSV files provided from the ACMS system. ACMS consists of several aircraft built-in sensors that traces the events generated by a sensor during a flight.

1 Sources

The scope of the project with respect to the sources is defined via the following constraints.

- The Data Warehouse: typically, for predicting an operation interruption, a good share of the data warehouse KPIs would be included as features. In our case, we will simplify the problem and consider only three KPIs: flight hours (FH), flight cycles (FC) and delayed minutes (DM).
  - We provide a functioning Data Warehouse meeting the first lab's requirements. This is available in the DW database, in FIB's PostgreSQL server:
    - The parameters are the same as the ones used to connect to AMOS, AIMS. Only the name of the database is **DW**.
    - Note that this Data Warehouse records *AircraftUtilization* and *LogBookReporting* at the day level (and not per month). This change is introduced to facilitate this lab.
- The ACMS system: we provide sensor data generated per flight. Since each aircraft has several subsystems, each of them with several sensors, we simplify the problem by focusing on the aircraft navigation subsystem (ATA code: **3453**) and assuming this subsystem has one single sensor. This data is gathered as follows:
  - The sensor generates an event every 5 minutes from take-off until landing, collecting the measurement of the sensor. This information is stored per flight in a single CSV file.
  - For this subsystem, we will thus work with CSV files.
  - You will find a .ZIP file (trainingData.zip) containing 376 CSV files in the resources directory of the code skeleton.

  In summary, the variables we are going to consider to predict if an aircraft is going to interrupt operations in the next 7 days are: the FH, FC and DM KPIs, and the aircraft navigation subsystem data collected as CSV files.

2 Processing

To implement the required data pipeline, you must create three data processing pipelines (i.e., management, analysis and evaluation) in order to predict the need for maintenance for an aircraft. These pipelines must be implemented in Spark using the PySpark library. The Python code developed should adhere to the instructions that follow.

2.1 Data Management Pipeline

This pipeline generates a matrix where the rows denote the information of an aircraft per day, and the columns refer to the FH, FC and DM KPIs, and the average measurement of the 3453 sensor. Thus, this pipeline must:

- Read the sensor measurements (extracted from the CSV files) related to a certain aircraft A and average it per day.
- Once you have the average measurement of the sensor per day, enrich it with the KPIs related to A from the Data Warehouse (at the same granularity level).
- Importantly, since we are going to use supervised learning (see the Data Analysis Pipeline below), we need to label each row with a label: either unscheduled maintenance or no maintenance predicted in the next 7 days for that flight.
- Generate a matrix with the gathered data and store it.

2.2 Data Analysis Pipeline

By using the data matrix created in the previous pipeline, we want to train a set of classifiers. In the literature, we can find many papers reporting that decision trees perform well for predicting transport delays, so one of the classifiers to be tested can be a Decision Tree. You can use the `spark.mllib` or the `spark.ml` package to train the classifiers. Thus the analysis pipeline must:

- Create two datasets (training and validation) and format them according to what is expected by the classifier algorithms.
- Create and store at least two classification models together with their corresponding hyperparameters and evaluation metrics over the validation sets. The evaluation metrics can be the traditional ones like predictive accuracy and recall.
  - You are recommended to use the MLflow[1] platform to manage the data analysis pipeline (e.g., to store/track the models, hyperparameters and evaluation metrics).
- Rank the models by their performance in the validation set (e.g., using predictive accuracy).
- Automatically deploy the model that performs best (i.e., set the model with the highest predictive accuracy as the default model to be used when performing predictions with new data).

2.3 Run-time Classifier Pipeline

Finally, once the models have been created, we must create a pipeline that, given a new record (<aircraft, day, FH, FC, DM, AVG(sensor)>), predicts if this aircraft is going to go for unscheduled maintenance. This pipeline must:

- Given an aircraft and a day, replicate the data management pipeline (i.e., extracts the KPIs from the DW and computes the average measurement of the sensor for that day from the available CSVs) and prepare the tuple to be inputted into the model.
- Classify the record and output maintenance / no maintenance.

[1] `https://mlflow.org`

Note 1: Even if the analytical aspects of the problem fall out of scope (i.e., we are not looking to find the absolute best model, but we want to try a couple of models and store the data associated with them), we do consider if you follow good practices when preparing the data to feed the model.

Note 2: Even if this lab is a simplified version of what would be required in a realistic case, it is still representative. In the following we list a set of characteristics that can be expected in a real scenario:

- Typically, the features used in the model are not decided beforehand. Instead, first a model is created and interpreted. Then, according to this interpretation, these pipelines are iteratively modified to consider new features and fine-tune the model. However, this part of the problem falls out of scope of this lab and we set the features beforehand.
- It is not a good practice to store the files generated by sensors in the local file system. Instead, a data lake (i.e., a database) should be deployed.
- A model might require many more features than the the ones considered for our models. However, from the data management point of view, it would entail more data flows consolidating variables into the matrix.
- The run-time classifier pipeline output would be implemented as an event in a dashboard. Thus, it would be one of the indicators / alarms / reports of a dashboard created with a visualization tool.

Deliverables

1. Three Python scripts corresponding to each pipeline described above. The three scripts must be included in a single zip file.
   - The Python code must include comments to facilitate the rationale you followed. At the header of each file, include an overall comment explaining WHAT are the steps implemented in the pipeline, and refer to these steps when explaining the Spark code in the subsequent comments.
   - The execution of the three pipelines should be facilitated. For instance, the code should not include absolute paths or fixed user credentials (e.g., they should be requested by the user or stored in configuration files).
2. A PDF file (max two A4 pages, 2.5cm margins, font size 12, inline space 1.15) with all assumptions made and justifying the decisions you made (if any).
   - Use one page to sketch the three pipelines at a higher abstraction level (i.e., you can group a couple of related Spark operations into one single box). Use the notation that you find more appropriate.
   - Use one page to elaborate on any assumption not stated in the lab statement but that you followed. This can be done for each one of the pipelines and refers to any specificity of your solution that should help the lecturer to understand the decisions you made in your code.

Assessment criteria

i) Conciseness of explanations (only first page will be considered in the evaluation)

ii) Understandability

iii) Coherence

iv) Soundness

Evaluation

- 60% Deliverables
- 40% Exercises related to the project done individually in the day of the final exam