Quick start with ML and Dagster

This article is a rapid introduction to Dagster using a small ML project. It is beginner friendly but might also suit more advanced programmers if they dont' know Dagster.

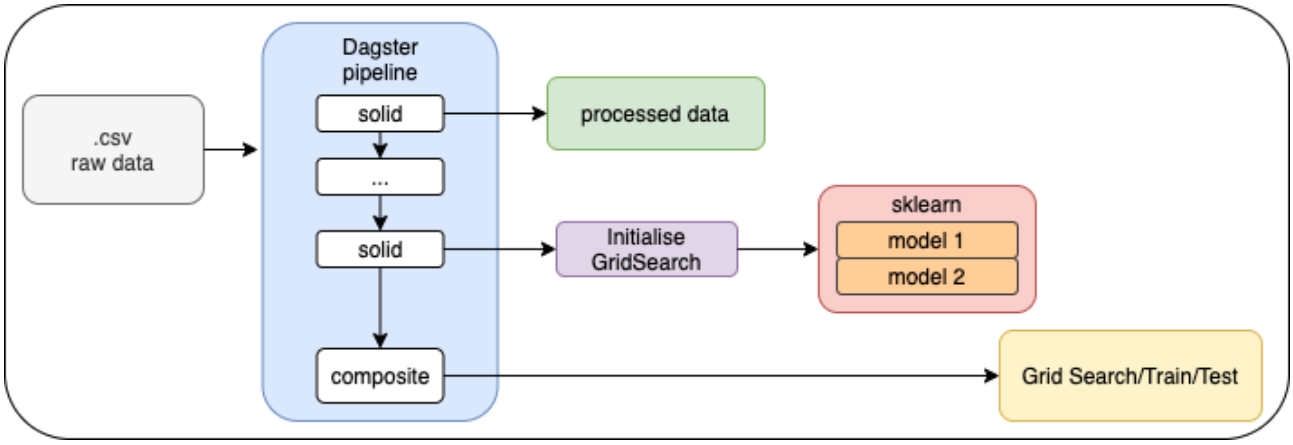
What is Dagster?

Dagster is a data orchestrator for machine learning, analytics, and ETL.
It lets you define **pipelines** in terms of the data flow between reusable, logical components, then test locally and run anywhere. With a unified view of pipelines and the assets they produce, Dagster can schedule and orchestrate Pandas, Spark, SQL, or anything else that Python can invoke. It makes testing easier and deploying faster.

To understand the fundamentals of Dagster, let's create a simple ML script.

The application

To keep it simple, let's create a text classifier with our old pal' Sklearn.
The code as well as further explanation to execute the code can be found at https://github.com/stephanBV/ML_with_DAGs



A solid is a unit of computation that yields a stream of events (similar to a function), while a composite is a collection of solids.

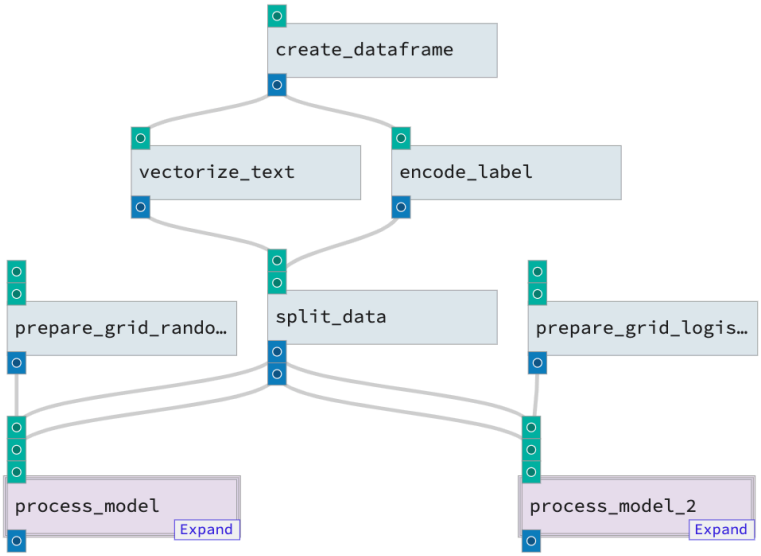
The script creates a single pipeline which:
-processes the data,
-searches for optimal parameters between a logistic regression and a random forest,
-train and test the model

The data

The raw data is a collection a comments from a social media platform, each comment has been labeled 'positive' or 'negative'.

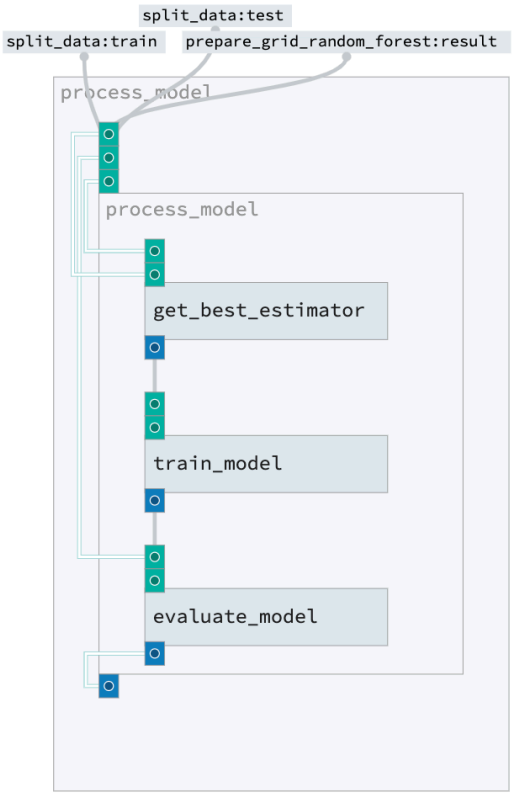
Launching Dagster

Once we launch the Dagster's UI, we get the following DAG:



We can see each **solid in light blue** and how they are connected to each other, as well as the **composites in light purple**.
NB: each solid also has an 'INFO" section which describes its inputs and outputs.

Furthermore, each composite has an **"Expand"** button, which let you see the set of solids it contains:



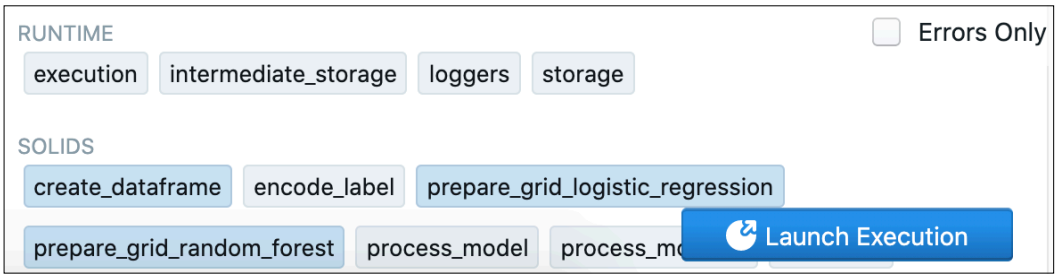
Here, you can see that the composite "process_model" contains the grid search, training and test solids. At the top, you can also see three inputs: result, train and test, which are the outputs of the solids "prepare_grid_random_forest", "split_data" with the train data and "split_data" with the test data.

Parameters can be in a YAML file in your repository or can be added to the **Playground** section, as follow:

```yaml
solids:
  create_dataframe:
    inputs:
      csv_path:
        value: "data/raw/comments_train.csv"
  prepare_grid_random_forest:
    inputs:
      key:
        value: 'rf'
      param_grid:
        value: {"n_estimators": [5, 10],
                "max_features": ["auto"],
                "max_depth": [1,2]}
  prepare_grid_logistic_regression:
    inputs:
      key:
        value: 'reg'
      param_grid:
        value: {"penalty":["l2","none"],
                "class_weight": ["balanced","None"]}
```

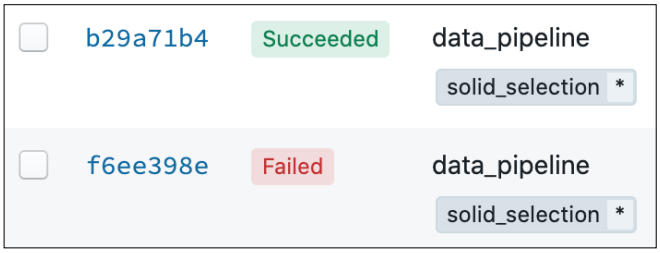
Playground also checks your parameters against your code in **real time** and will let you know instantly of any errors found.

If everything is okay, you will see at the bottom right of the window that the name of the solids with the parameters above are in a light-blue colour and the other solids are in grey colour.

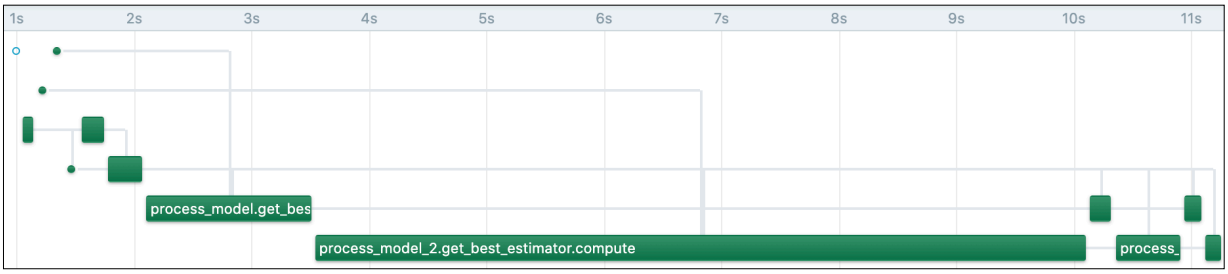


As long as nothing is red, you are good to go and you can **launch an execution.**

Once the pipeline has finished executing, the **Runs** section provides the list of all executions, as follow.



You can access the detail of all execution by clicking on its **id** (e.g. b29a71b4).
It provides a graphical representation of the execution duration,



-as well as a table which details all the steps that happened during the process. An important part on that table is the **step_logs** which is particularly useful, especially if you are running on DEBUG mode.

For example, the following is the output of the step_logs from the solid that tests the logistic regression.

```
INFO:root:
LogisticRegression(class_weight='None', random_state=31):
              precision    recall  f1-score   support

           0       0.82      0.81      0.82       121
           1       0.89      0.90      0.89       203

    accuracy                           0.86       324
   macro avg       0.86      0.85      0.85       324
weighted avg       0.86      0.86      0.86       324
```

Conclusion

To keep this article simple, only one pipeline was represented, although you can have as many pipelines as necessary. One could be specific to mine the data, another to process the data, another to test the code (with Pytest for e.g.), another could train, test, etc..
Dagster is an amazing tool to experiment, monitor and maintain your ML pipeline, It kind of make me think of an Open Source version of Azure ML Studio and Dataiku, but you get much more control over your project.

We only scratched the surface of Dagster. For more information, visit https://dagster.io/.