Practical MLOps: Do we need all the things?

Brian O. Bush

Principal Machine Learning Engineer

PyData 2022

SHIFT5

Intro

- Walk through traditional ML development (**issues**)
- The subset of MLOps that I believe is important
- Walk through a complete example of that subset with an open-source framework, Kedro

Resources for this tutorial

- Github repo: https://github.com/brianobush/pydata2022

Goals

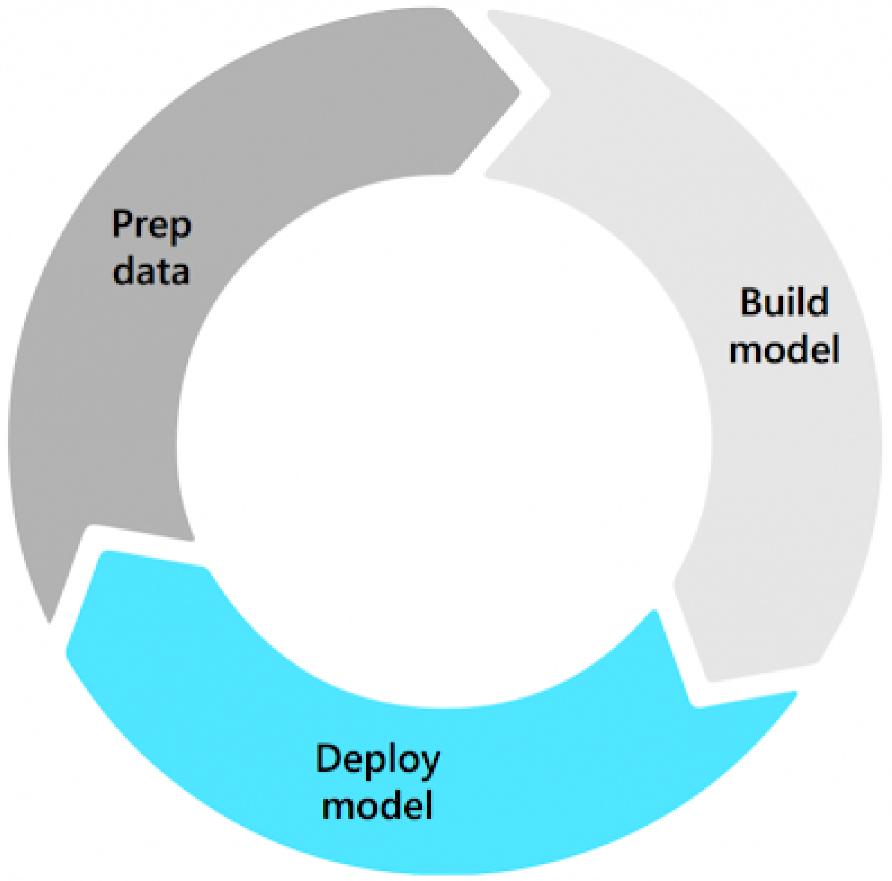
- **How to show ROI in ML endeavors**
    - get a handle on experiment execution
    - managing data source artifacts
    - model reproducibility
    - control ML development timelines

My ML Requirements (and thus biases)

- I develop ML-based technology for intrusion detection/anomaly detection in vehicles (planes, trains and tanks)
- Small team
- Embedded model deployment (embedded system with no internet)
- Long deployment/update cycle
- Require open-source framework(s)

Challenges In The Traditional ML Workflow

Compared to conventional software development, ML has unique challenges.



ML workflow: #1 Experiment Tracking

- ML is a metrics-driven process.
- In order to select the best model, multiple rounds of experiments are performed.
- For each experiment, one needs to track the metrics manually which increases the overall complexity.

ML workflow: #2 Reproducibility

- There is a lack of reproducibility as the experiments and the models produced (artifacts) are not tracked.

ML workflow: #3 Traceability

- The end product of the manual process is a single model rather than the whole pipeline.
- Do we know all of the steps to arrive at the same model?

ML workflow: #4 Data Versioning

- Data are not versioned. Similar to code, data also evolve.

ML workflow: #5 Model Versioning

No model versioning or model registry is available.

- Typically done by hand
- It is harder to reproduce a model from the past

ML workflow: #6 Oversight/Review

- Code reviews are typically not performed in ML.

ML workflow: #7 Testing

- Testing is usually missing, such as unit or integration tests

ML workflow: #8 Artifacts

- Collaboration between team members is a headache as it is difficult to share models and other artifacts.

ML workflow: #9 Logging

- Related to metrics, manual logging is an inefficient and error-prone process.

ML workflow: #10 Model Refreshing

A deployed model cannot easily be retrained, needed to address:

- concept drift
- accuracy improvements

As the process is manual, deploying a new model takes a lot of time and can result in less frequent releases

ML workflow: #11 Consistency

- Each project is organized differently. Hard for someone to jump into a project.
- Can be addressed by the Cookiecutter data science template or similar

Signs your traditional ML workflow isn't working

- Long ramp up times
- Technical debt (copy/pasting code; each project is a *snowflake*)
- Cannot easily reproduce previous models
- Scaling issues (people/process)

What is MLOps?

- Probably the most overused term in the AI/DS/ML industry
- In short, getting your models into production (and maintaining) them with the traditional ML workflow is **hard**
- MLOps defines an approach to deploy, maintain and operate ML models in production
- Doesn't have to be a enveloping as hyperscale companies (Uber,Google,Spotify)

Components of MLOps: Goals

Core components of MLOps that you need to implement (somehow):

- Data ingestion
- Pipeline and orchestration
    - data analysis, transforming, feature engineering, initial model development
- Model registry and experiment tracking
- Model deployment and serving
- Model monitoring

Components of MLOps: Reality

How I have addressed those core components of MLOps:

- Data ingestion (CSV, JSON, proprietary file-based blobs)
- Pipeline and orchestration (script+cron+kedro)
    - data analysis, transforming, feature engineering, initial model development
- Model registry and experiment tracking (github+mlflow)
- Model deployment and serving (script)
- Model monitoring (logging, prometheus)

Kedro and Terminology

- We are going to use an open-source MLOps platform to address some pain points
- Kedro is defined by a structure of nodes and pipelines.
- **Nodes** are the functions that perform any operations on the data.
- Sequencing of those nodes is a **pipeline**.

Problem Domain

- We will build an **anomaly detection pipeline** to identify anomalies in temperature values
- Use **isolation forest** as our machine learning model
- Get data here: https://github.com/brianobush/pydata2022/blob/main/temps.csv

Step 1: Starting Out with Kedro

- Create a new environment

```
Anaconda: conda create --name env python=3.7 -y

PyEnv: python3 -m venv env      (+activate environment)
```

- Activate your environment and Install required packages

```
pip install kedro kedro-viz numpy pandas scikit-learn
```

(You should also start versioning with `git init`; concepts not explored in detail here)

- Create a new Kedro project; I named mine `pydata2022`

```
kedro new
```

Step 2: Project Setup

- What did `kedro new` get us?
- We should see at least five directories:
  - **/data** - input, derived, output data
  - **/conf** - config files: data source, model params, etc.
  - **/src** - source code (pipeline steps, data processing, model training)
  - **/docs**
  - **/logs**

Step 3: Data Setup

- We first need to start organizing our data, enter the **data catalog**
- We will be editing `pydata2022/conf/base/catalog.yml`, below is our first entry:

```
temperature_database:
  type: pandas.CSVDataSet
  filepath: data/01_raw/temps.csv
```

- Copy the `temps.csv` file to `pydata2022/data/01_raw`
- another config file is `parameters.yml`, which we will visit later

Step 3: Data Setup, Validation Checkpoint

- Inside our project directory, let's make sure that we updated the data catalog correctly.

```
$ kedro catalog list

DataSets in '__default__' pipeline:
  Datasets not mentioned in pipeline:
    CSVDataSet:
    - temperature_database
```

Step 3: Data Setup Validation Checkpoint, Part 2

- We can also dive in and examine our data

```
$ kedro ipython
In [1]: x = catalog.load('temperature_database')
[11/07/22 01:10:57] INFO     Loading data from
'temperature_database' (CSVDataSet)...

In [2]: x.head()
Out[3]:
   2022-09-10T00:00:46   108
0  2022-09-10T00:05:22   102
1  2022-09-10T00:10:13   104
```

Step 4: Create Pipelines, Naming Best Practices

- **pre_processing**: initial cleaning data
- **data_engineering**: feature creation, train/test splits, etc
- **data_science**: ML modeling magic
- **metrics**: model evaluation

  Note: Whatever the convention, stick with it!

Step 4: Create Pipelines, Our First Node

- Let's create our first node in our pipeline, **data_engineering**

```
$ kedro pipeline create data_engineering

...

Pipeline 'data_engineering' was successfully created.

To be able to run the pipeline 'data_engineering', you will need to
add it to 'register_pipelines()' in
'pydata2022/src/pydata2022/pipeline_registry.py'.
```

Step 4: Create Pipelines, the data_science node

- Go ahead and create our data science node.

  ```
  $ kedro pipeline create data_science
  ```

Step 4: Create Pipelines, Validation Checkpoint

- In our project directory, you should see the pipelines in the `src` directory; you should see two directories

```
$ ls src/pydata2022/pipelines

data_engineering
data_science
```

Step 5: Building Out The Data Engineering Pipeline

- This is where we process/transform the data; see `supporting.py` on github.
- We add a function to split data into train/test sets; `src/pydata2022/pipelines/data_engineering/nodes.py`
- Import new function into pipeline and create node in pipeline; `src/pydata2022/pipelines/data_engineering/pipeline.py`
- Test our new node in the pipeline:

```
kedro run --node=node_train_test_split
...

kedro run
```

Step 5: Visualizing The Data Engineering Pipeline

- We should now be able to visualize what we have built. This should open up a web browser.

```
$ kedro viz
```

Step 6: Building Out The Data Science Pipeline

- We are limited to unsupervised techniques, as we have no labels in our dataset
- We will be using `isolation forest` to find outliers
- One of the most important parameters in this algorithm is the **contamination factor**
- We will add our model building routine to `src/pydata2022/pipelines/data_science/nodes.py`
- Then we import new our functions into pipeline and create two nodes in pipeline; `src/pydata2022/pipelines/data_science/pipeline.py`

Step 6: Side Note for Parameters

- In the Data Science Pipeline, you will note: `params:contamination_factor` in our pipeline definition; this allows for easy config based parameter tracking
- Open up parameters.yml in `conf/base/parameters.yml` and add the following line: `contamination_factor: 0.05`
- Better to add to parameters than to hard-code and fix later

Step 7: Registering All Pipelines

- This is the glue, where all pipelines are connected
- In `src/pydata2022/pipeline_registry.py` you will replace the existing `register_pipelines` code with a new one that brings both pipelines into play.
- Let's see what we built. Go to the terminal where you have been running commands and run:

  ```
  kedro viz
  ```

Step 8: Run the Pipeline

- We can now execute our defined pipeline, with:

  `kedro run`

- Remember you can still run subsets (nodes or pipelines), e.g.,

  ```
  kedro run --pipeline=data_engineering

  kedro run --node=split_data
  ```

  These can be very useful for debugging and analysis

What is next? Deploy!

- Once you have developed a working pipelines, you could deploy your solution with `kedro package`
- Generate your project documentation using `kedro build-docs`. Creates HTML documentation using Sphinx, so it includes your docstrings. Time to write some docs!

What Did Kedro Solve?

- **Reproducibility**: Can recreate steps of a workflow accurately and consistently
- **Modularity**: Code is self-contained and understandable; easy to test and modify
- **Maintainability**: Standard code templates that allow teammates to readily comprehend and maintain any project

What Did Kedro Solve? (Cont)

- **Versioning**: Tracking of data, configs, and machine learning model used in each pipeline run of every project
- **Documentation**: Clear and structured code information for easy reading and understanding.
- **Packaging**: Allows ML projects to be documented and shipped efficiently into production

How To Move Forward

- Pick an existing project and convert to Kedro
- Document the process
- The hardest part: **Starting**

Summary

- The approach shown is only a tiny problem, however, larger problems will have a similar structure
- We have shown a non-opinionated and straightforward method for doing *core* MLOps
- Take a look at `MLFlow` for model versioning. Integrates well with `Kedro`.

Resources

Note: I will put this talk and supporting materials on github.

- MLOps.org
- Kedro Github
- Kedro Docs
- Kedro-MLFlow; MLFlow integration improves versioning (with UI)
- Cookie Cutter Data Science

Questions/Comments

Thank you