IT UNIVERSITY OF COPENHAGEN
LARGE SCALE DATA ANALYSIS 2022

Assignment 3: ML Lifecycle

Tracking and reproducibility using MLflow

April 11, 2022

1 INTRODUCTION

In this assignment you will revisit the wind power forecasting system you made in Assignment 1, and use this as a case to go through some of the steps in the data analytics lifecycle (Figure 1.1).

You will do model selection by experimental evaluation and package your final model in a format that ensures reproducibility across platforms. You can reuse your preprocessing pipeline from Assignment 1 as the basis for your experiments.

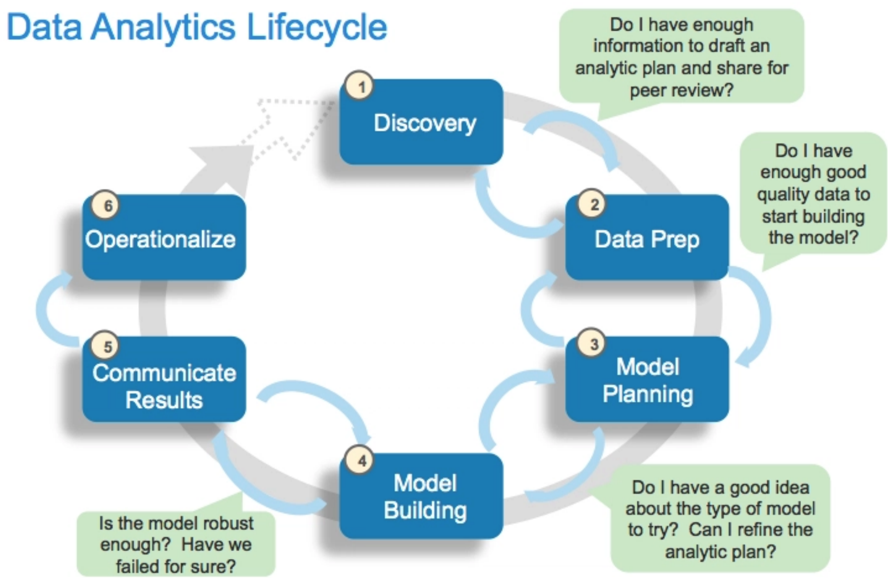

Figure 1.1: The data analytics life cycle. MLflow addresses tracking (2,3,4), reproducibility (4,5), and deployment (6)

2 MLflow

We will use MLflow, a set of open source tools for the machine learning lifecycle. It is a fairly new project, which is under rapid development.

MLflow covers three main areas: Experiment tracking, reproducibility, and model deployment. You will use MLflow Tracking quite a bit, MLflow Projects (reproducibility), and also MLflow Models (deployment).

Prior to working on this assignment, we highly recommend you go through the MLflow Tutorial in some detail.

2.1 Experiment tracking

When developing ML models, you go through a lot of experiments, trying many different combinations of models, hyper-parameters and feature extraction. This means you need to keep track of a lot of metrics, and how you created each metric. This is what MLflow Tracking is trying to solve.

In MLflow each *experiment* can consist of many *runs*, and in each run you can log *parameters*, *metrics*, *tags* and *artifacts*. To get a better explanation of this, visit the MLflow Tracking introduction

A simple experiment could look like this:

```
import mlflow
with mlflow.start_run():
    mlflow.log_param("Param one", 1)
    mlflow.log_metric("Accuracy", 0.5)
```

By default, MLflow will log your runs to the default experiment, and will save all your runs to your local file system in the folder `mlruns`. After running the example above, the directory structure could look like this:

```
mlruns/
    0/
      1f880fec49d64bffa1fdd4e7600f7c5b/
          artifacts/
          meta.yaml
          metrics/
              Accuracy
          params/
              Param one
          tags/
              mlflow.source.git.commit
              mlflow.source.name
              mlflow.source.type
              mlflow.user
      meta.yaml
```

The experiment id is `0`, the run id is `1f880fec49d64bffa1fdd4e7600f7c5b` and everything is just saved as text files. Try opening some of these files in your text editor. When running MLflow from within a git repository, the current commit is also saved as a tag, making it easier to recreate experiments.

MLflow also have experimental autologging features, which you are welcome to try out.

2.1.1 Azure

Microsoft Azure offers an interface for ML experiment tracking. You should all be eligible for free student credits, giving you some credits to experiment with.

You can follow these guides to get you started:

1. Setup your student account
2. Setup a VM with a public IP:

   - You should pick a region a little closer to home. Like Germany or Sweden.
   - In the Image choose Ubuntu Server 20.04.
   - You should pick a cheaper image. Like "B1s", which is only 60kr/month.
   - You can choose to use your existing public key instead of generating a new pair, if you wish.
   - You should allow port 22 (for SSH) and port 80 (HTTP), and RDP (3389) later.
3. Create Azure Machine Learning workspace
4. Go in the ML workspace you just created, download config.json life and store it locally together with your code.
5. Launch studio
6. Run your experiments locally (or on Azure VM, see section To the cloud! steps 2-5 to set up the VM to be able to run mlflow there) and you should be able to see the results in the Experiments tab on Microsoft Azure Machine Learning Studio.

2.1.2 Public tracking server

In addition to tracking your experiments on your local computer, you can also use a public tracking server. We have set up such a server at `http://training.itu.dk:5000/`. This tracking server stores runs in a PostgreSQL database instead of the local file system.

MLflow is not suitable as a competition framework, and is not really intended for many users. There is no authentication and results are not validated. This means that other users can delete your runs/experiments (possibly by mistake) and it is trivial to log fake metrics. Do not use the public tracking server for results that you cannot afford to lose.

So take it with a grain of salt.

2.2 Reproducibility

MLflow Projects is a standardized way to encapsulate an experiment in a reproducible manner. This is done by specifying all the dependencies as either a `conda` or `docker` environment. Here we will only talk about the `conda` approach.

An MLflow project consists of:

- The code you want to package, including the data for your experiments
- An environment file specifying the dependencies for the code. In this case a YAML file with the conda environment.
- An `MLproject` file specifying which environment file to use, and the entry points to the code.

We've made a small project that shows an example of polynomial regression, which we will use to show how MLflow projects work: `https://github.com/NielsOerbaek/PolyRegExample`

The main experiment, in which we simulate some data[1] and model it using different degrees of polynomial regression, is defined in `experiment.py`:

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from matplotlib import pyplot as plt
import numpy as np

import sys
num_samples = int(sys.argv[1]) if len(sys.argv) > 1 else 100

def get_ys(xs):
    signal = -0.1*xs**3 + xs**2 - 5*xs - 5
    noise = np.random.normal(0,100,(len(xs),1))
```

[1]The data is simulated using the polynomial function $f(x) = -0.1x^3 + x^2 - 5x + 5 + \epsilon$, where ϵ is Gaussian noise.

```
    return signal + noise

X = np.random.uniform(-20,20,num_samples).reshape((num_samples,1))
y = get_ys(X)

plt.scatter(X,y,label="data")

for degree in range(1,4):
    model = Pipeline([
        ("Poly", PolynomialFeatures(degree=degree)),
        ("LenReg", LinearRegression())
    ])
    model.fit(X,y)
    plotting_x = np.linspace(-20,20,num=50).reshape((50,1))
    preds = model.predict(plotting_x)
    plt.plot(plotting_x, preds, label=f"degree={degree}")

plt.legend()
plt.show()
```

The Conda environment for this experiment is specified in `PolyReg.yaml`:

```
name: PolyReg
channels:
  - defaults
dependencies:
  - scikit-learn>0.23
  - numpy>1.19
  - pip>20
  - python=3.8
  - pip:
    - mlflow
    - matplotlib>3
```

Finally, the `MLproject`-file specifies how to run the project:

```
name: PolyReg

conda_env: PolyReg.yaml

entry_points:
  main:
    parameters:
      num_samples: {type: int, default: 100}
    command: "python experiment.py {num_samples}"
```

With these three things in place, you can run the experiment using `mlflow run <path to project>`. So if the project is located on your computer, you can navigate to the directory and do:

```
mlflow run .
```

Because this project is hosted as a git repository, you can simply do:

```
mlflow run https://github.com/NielsOerbaek/PolyRegExample.git
```

This will fetch the project, resolve the environment, and run the main entry point with the default parameters.

If you want to run the experiment with 500 samples, instead of the default 100, you can do:

```
mlflow run https://github.com/NielsOerbaek/PolyRegExample.git -P
   num_samples=500
```

2.3 Deployment

Once you have a model you are satisfied with, you can log and deploy it using MLflow Models model format. There's detailed descriptions of the deployment options in the documentation.

We will not spend too much energy of this, but it you are interested, have a look at this repo: https://github.com/NielsOerbaek/caiso-mlflow. This is an MLflow Project that trains a CAISO model for electricity load forecasting and saves it to the local filesystem. The saved model can then be distributed and deployed using MLflow Models.

To use the saved artifacts, we need to clone the repo to your filesystem before running (otherwise the model will just be saved in a temporary folder).

```
git clone https://github.com/NielsOerbaek/caiso-mlflow
cd caiso-mlflow
mlflow run .
mlflow models serve -m model
```

The model will now be served on your local machine. You can then query your model by opening another terminal and running:

```
curl http://127.0.0.1:5000/invocations -H 'Content-Type: application/json'
    -d '{
  "columns": ["Time"],
  "data": [["2020-11-14T20:00:00"]]
}'
```

You will then get an answer like `[16.07111111111111]`, meaning that your model thinks that the electricity demand in Orkney at 8 PM on Saturday the 14th of November 2020 will be 16.07 MW.

2.3.1 To the cloud!

Deploying to your own machine might seem a bit roundabout. The interesting thing comes when deploying to a server and can be queried by many users.

You can look into deploying your model to a cloud-based virtual machine with a public IP address. One such option is to use Microsoft Azure. MLflow has methods for deploying directly to Azure, but we've had mixed experiences with it. Again, have a look at the documentation.

If you already have a VM on Azure (if not, follow section Azure steps 1-2) you can ssh into it and serve your model:

1. Push your MLflow projet to Github if you develop your project locally. In case you did your MLflow stuff (running experiments/training model) on Azure VM, you can skip steps 3-6.
2. ssh into your VM on Azure `ssh <username>@<Public IP address>`. You can find VM Public IP address in the Overview.
3. Get miniconda on the VM:

   ```
   wget https://repo.anaconda.com/miniconda/Miniconda3-py39_4.10.3-Linux-
       x86_64.sh
   bash Miniconda3-py39_4.10.3-Linux-x86_64.sh
   ```

4. Exit VM and ssh again so that conda enviroment is activated.
5. Get MLflow:

   ```
   conda install -c conda-forge mlflow
   ```

6. Get your model from Github (if you are using our example MLflow project Caiso you need to rename MLProject file to MLproject):

   ```
   git clone https://github.com/NielsOerbaek/caiso-mlflow
   cd caiso-mlflow

   #rename MLproject file if it is named MLProject
   mv MLProject MLproject
   ```

7. Open port 5000[2]. Skip Create a network security group, go to your Azure Resource group and select existing Network security group.

[2] Port 5000 is default for MLflow

8. Serve the model:

```
mlflow run .
mlflow models serve -m model -h 0.0.0.0 -p 5000
```

9. Query served model from your local machine:

```
curl http://<Public IP address>:5000/invocations -H 'Content-Type:
    application/json' -d '{
  "columns": ["Time"],
  "data": [["2020-11-14T20:00:00"]]
}'
```

You will then get an answer like `[16.071111111111111]`, meaning that your model thinks that the electricity demand in Orkney at 8 PM on Saturday the 14th of November 2020 will be 16.07 MW.

You might also want to host a local tracking server on the same VM, and possibly a database or a S3-compatible object store. To start long-running processes that don't stop when you terminate your ssh connection, have a look at the nohup command. To install new services (like MinIO), you might want to use docker. Think about how these steps could be automated.

3 Data

For this assignment we have frozen the dataset to make comparisons easier. This means that the data will not be fetched from the influx database, but simply loaded from the attached json-file. The json file can be loaded into a pandas dataframe by using:

```
df = pd.read_json("path/to/file.json", orient="split")
```

The data format is the same as for Assignment 1; the only difference being that the `generation` and `wind` dataframe have been joined by an outer join. This you only need to load a single dataframe as your dataset.

The attached dataset covers 180 days of data. Below is the output of `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 254967 entries, 2020-05-15 12:55:00 to 2020-11-11 12:54:00
Data columns (total 7 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   ANM           254967 non-null  float64
 1   Non-ANM       254967 non-null  float64
 2   Total         254967 non-null  float64
 3   Direction     1379 non-null    object
 4   Lead_hours    1379 non-null    float64
 5   Source_time   1379 non-null    datetime64[ns]
 6   Speed         1379 non-null    float64
dtypes: datetime64[ns](1), float64(5), object(1)
memory usage: 15.6+ MB
```

3.1 Evaluation

The template file already comes with scaffolding to do cross validation of your models. This will be the basis for your metrics. You do not need to do an additional train/test split.

4 REQUIREMENTS AND HAND-IN

4.1 System requirements

For this assignment you should do model evaluation and selection using `mlflow` in a system that:

- Reads the data from a JSON file
- Trains a linear regression model and a model of your choice
- Implements cross-validation with different number of splits
- Tracks the evaluation errors for each model and cross-validation parameter.

Based on the evaluation errors, you should choose the model with the best overall performance, package the experiment as an MLflow Project, or save the model in the MLflow Model format, and deploy it on Azure.

We encourage you to experiment with both the preprocessing, feature extraction and hyper-parameters. Here are some examples of experiments you could do:

- Adding polynomial features (trying out different degrees)
- Upsampling/downsampling data
- Radian/ordinal/one-hot encoding of wind direction
- Set of features (what if you only include wind speed?)
- Hyper-parameters for the model of your choice

4.2 Hand-in

You should hand in a report describing:

1. Your choice of models and evaluation metrics
2. How the evaluation errors change in terms of the cross-validation parameters
3. Your choice of best performing model
4. The advantages of packaging the experiments/models in the MLflow formats and a comparison with other reproducibility options.

You might find it easier to discuss the evaluation metrics by plotting them as a function of the model and cross-validation parameters. Consider using some of the error metrics, statistics and visualisation methods described in the lectures. You should discuss the evaluation errors in relation to the generalisation power of the models. When describing the advantages of packaging models, make sure you highlight the key features of reproducible ML systems.

4.3 Suggested reading and useful links

- Pages 75-84 of Hands-On Machine Learning.
- MLflow Documentation
- Model evaluation in sklearn
- matplotlib library for visualisation

5 Code Template

```python
import pandas as pd
import mlflow

## NOTE: You can use Microsoft Azure Machine Learning Studio for experiment tracking. Follow assignment description and uncomment below for that (you might also need to pip azureml (pip install azureml-core):
#from azureml.core import Workspace
#ws = Workspace.from_config()
#mlflow.set_tracking_uri(ws.get_mlflow_tracking_uri())

## NOTE: Optionally, you can use the public tracking server.  Do not use it for data you cannot afford to lose. See note in assignment text. If you leave this line as a comment, mlflow will save the runs to your local filesystem.

# mlflow.set_tracking_uri("http://training.itu.dk:5000/")

# TODO: Set the experiment name
mlflow.set_experiment("<ITU Username> - <Descriptive experiment name>")

# Import some of the sklearn modules you are likely to use.
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Start a run
# TODO: Set a descriptive name. This is optional, but makes it easier to keep track of your runs.
with mlflow.start_run(run_name="<descriptive name>"):
    # TODO: Insert path to dataset
    df = pd.read_json("path/to/dataset.json", orient="split")

    # TODO: Handle missing data

    pipeline = Pipeline([
        # TODO: You can start with your pipeline from assignment 1
    ])

    # TODO: Currently the only metric is MAE. You should add more. What other metrics could you use? Why?
    metrics = [
        ("MAE", mean_absolute_error, []),
    ]

    X = df[["Speed","Direction"]]
    y = df["Total"]

    number_of_splits = 5

    #TODO: Log your parameters. What parameters are important to log?
    #HINT: You can get access to the transformers in your pipeline using `pipeline.steps`

    for train, test in TimeSeriesSplit(number_of_splits).split(X,y):
        pipeline.fit(X.iloc[train],y.iloc[train])
        predictions = pipeline.predict(X.iloc[test])
        truth = y.iloc[test]
```

```python

        from matplotlib import pyplot as plt
        plt.plot(truth.index, truth.values, label="Truth")
        plt.plot(truth.index, predictions, label="Predictions")
        plt.show()

        # Calculate and save the metrics for this fold
        for name, func, scores in metrics:
            score = func(truth, predictions)
            scores.append(score)

    # Log a summary of the metrics
    for name, _, scores in metrics:
            # NOTE: Here we just log the mean of the scores.
            # Are there other summarizations that could be interesting?
            mean_score = sum(scores)/number_of_splits
            mlflow.log_metric(f"mean_{name}", mean_score)
```