Catch Me If You Can: Keeping up with ML Models in Production

Shreya Shankar

May 2021

Outline

Introductions

- BS and MS from Stanford Computer Science
- First machine learning engineer at Viaduct, an applied ML startup
    - Working with terabytes of time series data
    - Building infrastructure for large-scale machine learning and data analytics
    - Responsibilities spanning recruiting, SWE, ML, product, and more

Academic and industry ML are different

- Most academic ML efforts are focused on training techniques
  - Fairness
  - Generalizability to test sets with unknown distributions
  - Security
- Industry places emphasis on Agile working style
- In industry, we want to train few models but do *lots* of inference
- What happens beyond the validation or test set?

The depressing truth about ML IRL

- 87% of data science projects don't make it to production[1]
- Data in the "real world" is not necessarily clean and balanced, like canonical benchmark datasets (ex: ImageNet)
- Data in the "real world" is always changing
- Showing high performance on a fixed train and validation set $\neq$ consistent high performance when that model is deployed

[1] https://venturebeat.com/2019/07/19/why-do-87-of-data-science-projects-never-make-it-into-production/

Today's talk

- *Not* about how to improve model performance on a validation or test set
- *Not* about feature engineering, machine learning algorithms, or data science
- Discussing:
    - Case study of challenges faced post-deployment of models
    - Showcase of tools to monitor production pipelines and enable Agile teams of different stakeholders to quickly identify performance degradation
- Motivating when to retrain machine learning models in production

General problem statement

- Common to experience *performance drift* after a model is deployed
- Unavoidable in many cases
  - Time series data
  - High-variance data
- Intuitively, a model and pipeline should not be *stale*
  - Want the pipeline to reflect most recent data points
- Will simulate performance drift with a toy task — predicting whether a taxi rider will give their driver a high tip or not

Case study description

- Using publicly available NYC Taxicab Trip Data (data in public S3 bucket `s3://nyc-tlc/trip data/`)
- For this exercise, we will train and "deploy" a model to predict whether the user gave a large tip
- Using Python, Prometheus, Grafana, mlflow, mltrace
- Goal of this exercise is to demonstrate what can go wrong post-deployment and how to troubleshoot bugs
    - Diagnosing failure points in the pipeline is challenging for Agile teams when pipelines are very complex!

Dataset description

- Simple toy example using the publicly available NYC Taxicab Trip Data[2]
- Tabular dataset where each record corresponds to one taxi ride
  - Pickup and dropoff times
  - Number of passengers
  - Trip distance
  - Pickup and dropoff location zones
  - Fare, toll, and tip amounts
- Monthly files from Jan 2009 to June 2020
- Each month is about 1 GB of data $\rightarrow$ over 100 GB

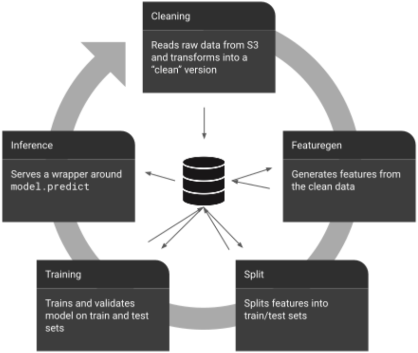
[2] https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Learning task

- Let X be the train covariate matrix (features) and y be the train target vector (labels)
  - X_i represents an n-dim feature vector corresponding to the ith ride and $y_i \in [0, 1]$ where 1 represents the rider tipping more than 20% of the total fare
- Want to learn classifier f such that $f(X_i) \approx y_i$ for all $X_i \in X$
- We do not want to *overfit*, meaning $F_1(X_{train}) \approx F_1(X_{test})$
- We will use a `RandomForestClassifier` with basic hyperparameters
  - `n_estimators=100, max_depth=10`

Toy ML Pipeline

Code lives in `toy-ml-pipeline`[3], an example of an end-to-end pipeline for our high tip prediction task.



[3] `https://github.com/shreyashankar/toy-ml-pipeline`

Toy ML pipeline utilities

- S3 bucket for intermediate storage
- io library
  - dumps versioned output of each stage every time a stage is run
  - load latest version of outputs so we can use them as inputs to another stage
- Serve predictions via Flask application

```
global mw
mw = models.RandomForestModelWrapper.load('training/
                                        models')

@app.route('/predict', methods=['POST'])
def predict():
    req = request.get_json()
    df = pd.read_json(req['data'])
    df['prediction'] = mw.predict(df)
    result = {
        'prediction': df['prediction'].to_list()
    }
    return jsonify(result)
```

ETL

- Cleaning stage reads in raw data for each month and removes:
  - Rows with timestamps that don't fall within the month range
  - Rows with $0 fares
- Featuregen stage computes the following features:
  - Trip-based: passenger count, trip distance, trip time, trip speed
  - Pickup-based: pickup weekday, pickup hour, pickup minute, work hour indicator
  - Basic categorical: pickup location code, dropoff location code, type of ride

Offline training and evaluation

- Train on January 2020, validate on February 2020
- We will measure F_1 score for our metric
  - precision $= \frac{\text{number of true positives}}{\text{number of records predicted to be positive}}$
  - recall $= \frac{\text{number of true positives}}{\text{number of positives in the dataset}}$
  - $F_1 = \frac{2*\text{precision}*\text{recall}}{\text{precision}+\text{recall}}$
  - Higher F_1 score is better, want to have low false positives and false negatives
- Steps
  1. Split the output of the `featuregen` stage into train and validation sets
  2. Train and validate the model in a notebook
  3. Train the model for production in another notebook

Train and test split code snippet

```
train_month = '2020_01'
test_month = '2020_02'

# Load latest features for train and test sets
train_df = io.load_output_df(os.path.join('features',
                                          train_month))
test_df = io.load_output_df(os.path.join('features',
                                         test_month))

# Save train and test sets
print(io.save_output_df(train_df, 'training/files/train'))
print(io.save_output_df(test_df, 'training/files/test'))
```

Output

```
s3://toy-applied-ml-pipeline/dev/training/files/train/20210501-165847.pq
s3://toy-applied-ml-pipeline/dev/training/files/test/20210501-170136.pq
```

Training code snippet

```
feature_columns = [
    'pickup_weekday', 'pickup_hour',
    'pickup_minute', 'work_hours',
    'passenger_count', 'trip_distance',
    'trip_time', 'trip_speed', 'PULocationID',
    'DOLocationID', 'RatecodeID'
]
label_column = 'high_tip_indicator'
model_params = {'max_depth': 10}

# Create and train model
mw = models.RandomForestModelWrapper(feature_columns=
                                     feature_columns, model_params=
                                     model_params)
mw.train(train_df, label_column)
```

Training and test set evaluation

```
train_score = mw.score(train_df, label_column)
test_score = mw.score(test_df, label_column)
mw.add_metric('train_f1', train_score)
mw.add_metric('test_f1', test_score)

print('Metrics:')
print(mw.get_metrics())
```

Output

Metrics:
'train_f1': 0.7310504153094398, 'test_f1': 0.735223195868965

Train the production model

- Simulate deployment in March 2020 (COVID-19 incoming!)
- In practice, train models on different windows/do more thorough data science
- Here, we will just train a model on February 2020 with the same parameters that we explored before

```
train_df = io.load_output_df(os.path.join('features', '
                                          2020-02'))
prod_mw = models.RandomForestModelWrapper(
                                          feature_columns=
                                          feature_columns,
                                          model_params=model_params)
prod_mw.train(train_df, label_column)
train_score = prod_mw.score(train_df, label_column)
prod_mw.add_metric('train_f1', train_score)
prod_mw.save('training/models', dev=False)
```

Deploy model

- Flask server that runs `predict` on features passed in via request
- Separate file `inference.py` that repeatedly sends requests containing features
- Logging metrics in the Flask app via Prometheus
- Visualizing metrics via Grafana
- Demo

Inspect F_1 scores at the end of every day

Output

	day	rolling_f1	daily_f1
178123	1	0.576629	0.576629
370840	2	0.633320	0.677398
592741	3	0.649983	0.675877
...			
1514827	28	0.659934	0.501860
1520358	29	0.659764	0.537860
1529847	30	0.659530	0.576178

Deeper dive into March F_1 scores

	day	rolling_f1	daily_f1
178123	1	0.576629	0.576629
370840	2	0.633320	0.677398
592741	3	0.649983	0.675877
...			
1507234	27	0.660228	0.576993
1514827	28	0.659934	0.501860
1520358	29	0.659764	0.537860
1529847	30	0.659530	0.576178

- Large discrepancy between rolling metric and daily metric
- What you monitor is important – daily metric significantly drops towards the end
- Visible effect of COVID-19 on the data

Evaluating the same model on following months

Output

2020-04
F1: 0.5714705472990737
2020-05
F1: 0.5530868473460906
2020-06
F1: 0.5967621469282887

Performance gets significantly worse! We will need to retrain models periodically.

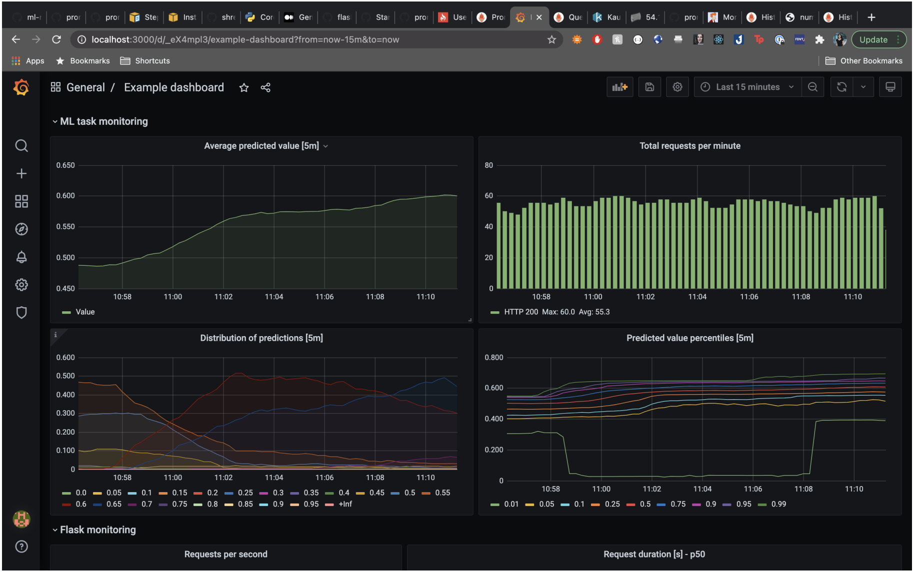
Challenge: lag

- In this example we are simulating a "live" deployment on historical data, so we have no lag
- In practice there are many types of lag
  - Feature lag: our system only learns about the ride well after it has occurred
  - Label lag: our system only learns of the label (fare, tip) well after the ride has occurred
- The evaluation metric will inherently be lagging
- We might not be able to train on the most recent data

Challenge: distribution shift

- Data "drifts" over time, and models will need to be retrained to reflect such drift
- Open question: how often do you retrain a model?
  - Retraining adds complexity to the overall system (more artifacts to version and keep track of)
  - Retraining can be expensive in terms of compute
  - Retraining can take time and energy from people
- Open question: how do you know when the data has "drifted?"

Demo


localhost:3000/d/_eX4mpl3/example-dashboard?from=now-15m&to=now
General / Example dashboard
Last 15 minutes
ML task monitoring
Average predicted value [5m]
Total requests per minute
HTTP 200 Max: 60.0 Avg: 55.3
Distribution of predictions [5m]
Predicted value percentiles [5m]
Flask monitoring
Requests per second
Request duration [s] - p50

How do you know when the data has "drifted?"

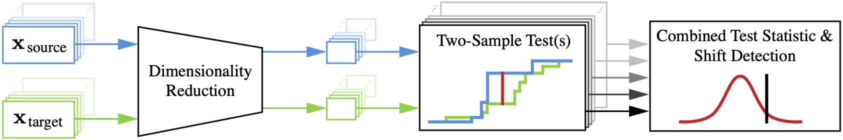


Figure 1: Our pipeline for detecting dataset shift. Source and target data is fed through a dimensionality reduction process and subsequently analyzed via statistical hypothesis testing. We consider various choices for how to represent the data and how to perform two-sample tests. [4]

[4] Rabanser, Günnemann, and Lipton, *Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift*.

Challenge: quantifying distribution shift

- We only have 11 features, so we'll skip the dimensionality reduction step
- "Multiple univariate testing seem to offer comparable performance to multivariate testing" (Rabanser et al.)
  - Maybe this is specific to their MNIST and CIFAR experiments
  - Regardless, we will employ multiple univariate testing
- For each feature, we will run a 2-sided Kolmogorov-Smirnov test
  - Continuous data, non-parametric test
  - Compute the largest difference

$$Z = \sup_z |F_p(z) - F_q(z)|$$

where F_p and F_q are the empirical CDFs of the data we are comparing

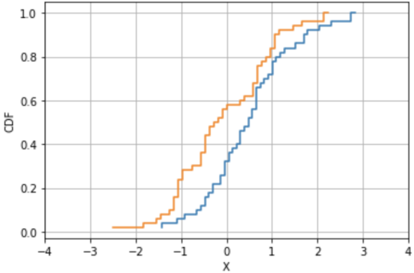
Kolmogorov-Smirnov test made simple

- For each feature, we will run a 2-sided Kolmogorov-Smirnov test
  - Continuous data, non-parametric test
  - Compute the largest difference

$$Z = \sup_z |F_p(z) - F_q(z)|$$

where F_p and F_q are the empirical CDFs of the data we are comparing

- Example comparing two random normally distributed PDFs



How do you know when the data has "drifted?"

Comparing January 2020 and February 2020 datasets using the KS test:

Output

	feature	statistic	p_value
0	pickup_weekday	0.046196	0.000000e+00
2	work_hours	0.028587	0.000000e+00
6	trip_time	0.017205	0.000000e+00
7	trip_speed	0.035415	0.000000e+00
1	pickup_hour	0.009676	8.610133e-258
5	trip_distance	0.005312	5.266602e-78
8	PULocationID	0.004083	2.994877e-46
9	DOLocationID	0.003132	2.157559e-27
4	passenger_count	0.002947	2.634493e-24
10	RatecodeID	0.002616	3.047481e-19
3	pickup_minute	0.000702	8.861498e-02

How do you know when the data has "drifted?"

- For our "offline" train and evaluation sets (January and February), we get *extremely* low p-values!
- This method can have a high "false positive rate"
  - In my experience, this method has flagged distributions as "significantly different" more than I want it to
  - In the era of "big data" (we have millions of data points), p-values are not useful to look at[5]

[5] Lin, Lucas, and Shmueli, "Too Big to Fail: Large Samples and the p-Value Problem".

How do you know when the data has "drifted?"

You don't really know.

An unsatisfying solution is to retrain the model to be as fresh as possible.

Types of production ML bugs

- Data changed or corrupted
- Data dependency / infra issues
- Logical error in the ETL code
- Logical error in retraining a model
- Logical error in promoting a retrained model to inference step
- Change in consumer behavior
- Many more

Why production ML bugs are hard to find and address

- ML code fails silently (few runtime or compiler errors)
- Little or no unit or integration testing in feature engineering and ML code
- Very few (if any) people have end-to-end visibility on an ML pipeline
- Underdeveloped monitoring and tracing tools
- So many artifacts to keep track of, especially if you retrain your models

What to monitor?

- Agile philosophy: "Continuous Delivery." Monitor proactively!
- Especially hard when there are no labels
- First-pass monitoring
  - Model output distributions
  - Averages, percentiles
- More advanced monitoring
  - Model input (feature) distributions
  - ETL intermediate output distributions
  - Can get tedious if you have many features or steps in ETL
- Still unclear how to quantify, detect, and act on

Prometheus monitoring for ML

Selling points	Pain points
Easy to call `hist.observe(val)`	User (you) needs to define the histogram buckets
Easy to integrate with Grafana	Metric naming, tracking, and visualization clutter
Many applications already use Prometheus	PromQL is not very intuitive for ML-based tasks

PromQL for ML monitoring

Question	Query
Average output [5m]	`sum(rate(histogram_sum[5m])) / sum(rate(histogram_count[5m]))`, time series view
Median output [5m]	`histogram_quantile(0.5, sum by (job, le) (rate(histogram_bucket[5m])))`, time series view
Probability distribution of outputs [5m]	`sum(rate(histogram_bucket[5m])) by (le)`, heatmap view[6]

[6] Jordan, *A simple solution for monitoring ML systems.*

Retraining regularly

Suppose we retrain our model on a weekly basis.

- Use MLFlow Model Registry[7] to keep track of which model is the best model
- Alternatively, can also manage versioning and a registry ourselves (what we do now)
- At inference time, pull the latest/best model from our model registry

Now, we need some tracing[8]...

[7] `https://www.mlflow.org/docs/latest/model-registry.html`

[8] Hermann and Del Balso, *Scaling Machine Learning at Uber with Michelangelo*.

mltrace

- Coarse-grained lineage and tracing
- Designed specifically for complex data or ML pipelines
- Designed specifically for Agile multidisciplinary teams
- Alpha release[9] contains Python API to log run information and UI to view traces for outputs

[9] `https://github.com/loglabs/mltrace`

mltrace design principles

- Utter simiplicity (user knows everything going on)
- No need for users to manually set component dependencies – the tool should detect dependencies based on I/O of a component's run
- API designed for both engineers and data scientists
- UI designed for people to help triage issues **even if they didn't build the ETL or models themselves**
    - When there is a bug, whose backlog do you put it on?
    - Enable people who may not have developed the model to investigate the bug

mltrace concepts

- Pipelines are made of components (ex: cleaning, featuregen, split, train)
- In ML pipelines, **different** artifacts are produced (inputs and outputs) when the **same** component is run more than once
- `Component` and `ComponentRun` objects[10]
- Decorator interface similar to Dagster "solids"[11]
  - User specifies component name, input and output variables
- Alternative Pythonic interface similar to MLFlow tracking[12]
  - User creates instance of `ComponentRun` object and calls `log_component_run` on the object

[10] https://mltrace.readthedocs.io/en/latest

[11] https://docs.dagster.io/concepts/solids-pipelines/solids

[12] https://www.mlflow.org/docs/latest/tracking.html

mltrace integration example

```python
def clean_data(raw_data_filename: str, raw_df: pd.DataFrame,
                                    month: str, year: str,
                                   component: str) -> str:
    first, last = calendar.monthrange(int(year), int(month))
    first_day = f'{year}-{month}-{first:02d}'
    last_day = f'{year}-{month}-{last:02d}'
    clean_df = helpers.remove_zero_fare_and_oob_rows(
        raw_df, first_day, last_day)

    # Write "clean" df to s3
    output_path = io.save_output_df(clean_df, component)

    return output_path
```

mltrace integration example

```
@register('cleaning', input_vars=['raw_data_filename'],
                                 output_vars=['output_path'])
def clean_data(raw_data_filename: str, raw_df: pd.DataFrame,
                                  month: str, year: str,
                                 component: str) -> str:
    first, last = calendar.monthrange(int(year), int(month))
    first_day = f'{year}-{month}-{first:02d}'
    last_day = f'{year}-{month}-{last:02d}'
    clean_df = helpers.remove_zero_fare_and_oob_rows(
        raw_df, first_day, last_day)

    # Write "clean" df to s3
    output_path = io.save_output_df(clean_df, component)

    return output_path
```

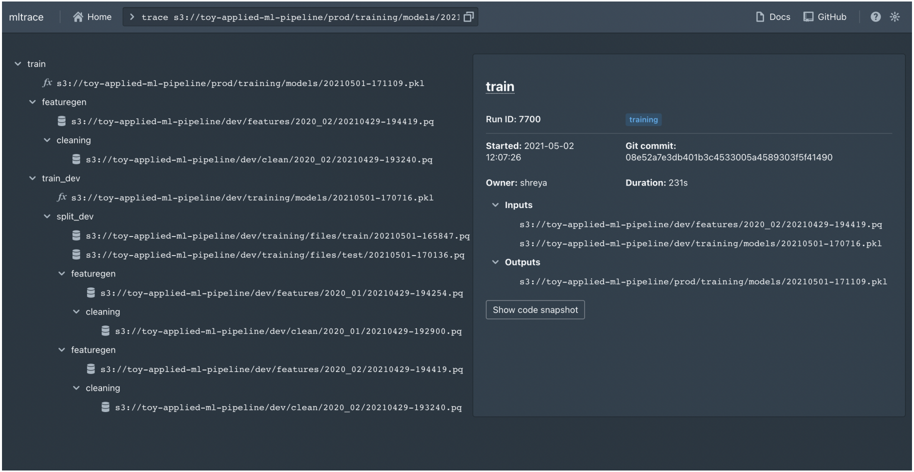
mltrace integration example

```
from mltrace import create_component, tag_component,
                              register

create_component('cleaning', 'Cleans the raw data with basic
                               OOB criteria.', 'shreya')
tag_component('cleaning', ['etl'])

# Run function
raw_df = io.read_file(month, year)
raw_data_filename = io.get_raw_data_filename('2020', '01')
output_path = clean_data(raw_data_filename, raw_df, '01', '
                              2020', 'clean/2020_01')
print(output_path)
```

mltrace UI demo



mltrace immediate roadmap[13]

- UI feature to easily see if a component is stale
  - There are newer versions of the component's outputs
  - The latest run of a component happened weeks or months ago
- CLI to interact with mltrace logs
- Scala Spark integrations (or REST API for logging)
- Prometheus integrations to easily monitor outputs
- Possibly other MLOps tool integrations

[13] Please email shreyashankar@berkeley.edu if you're interested in contributing!

Talk recap

- Introduced an end-to-end ML pipeline for a toy task
- Demonstrated performance drift or degradation when a model was deployed via Prometheus metric logging and Grafana dashboard visualizations
- Showed via statistical tests that it's hard to know when exactly to retrain a model
- Discussed challenges around continuously retraining models and maintaining ML production pipelines
- Introduced mltrace, a tool developed for ML pipelines that performs coarse-grained lineage and tracing

Areas of future work in MLOps

- Only scratched the surface with challenges discussed today
- Many more of these problems around deep learning
  - Embeddings as features – if someone updates the upstream embedding model, do all the data scientists downstream need to immediately change their models?
  - "Underspecified" pipelines can pose threats[14]
- How to enable anyone to build dashboards or monitor pipelines?
  - ML people know what to monitor, infra people know how to monitor
  - We should strive to build tools to allow engineers and data scientists to be more self sufficient

[14] D'Amour et al., *Underspecification Presents Challenges for Credibility in Modern Machine Learning*.

Miscellaneous

- Code for this talk: `https://github.com/shreyashankar/debugging-ml-talk`
- Toy ML Pipeline with mltrace, Prometheus, and Grafana: `https://github.com/shreyashankar/toy-ml-pipeline/tree/shreyashankar/db`
- mltrace: `https://github.com/loglabs/mltrace`
- Contact info
    - Twitter: @sh_reya
    - Email: shreyashankar@berkeley.edu
- Thank you!

References

D'Amour, Alexander et al. *Underspecification Presents Challenges for Credibility in Modern Machine Learning*. 2020. arXiv: 2011.03395 [cs.LG].

Hermann, Jeremy and Mike Del Balso. *Scaling Machine Learning at Uber with Michelangelo*. 2018. URL: https://eng.uber.com/scaling-michelangelo/.

Jordan, Jeremy. *A simple solution for monitoring ML systems.* 2021. URL: https://www.jeremyjordan.me/ml-monitoring/.

Lin, Mingfeng, Henry Lucas, and Galit Shmueli. "Too Big to Fail: Large Samples and the p-Value Problem". In: *Information Systems Research* 24 (Dec. 2013), pp. 906–917. DOI: 10.1287/isre.2013.0480.

Rabanser, Stephan, Stephan Günnemann, and Zachary C. Lipton. *Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift*. 2019. arXiv: 1810.11953 [stat.ML].

Feedback

Your feedback is important to us. Don't forget to rate and review the sessions.