

ML: страх и ненависть в продакшне

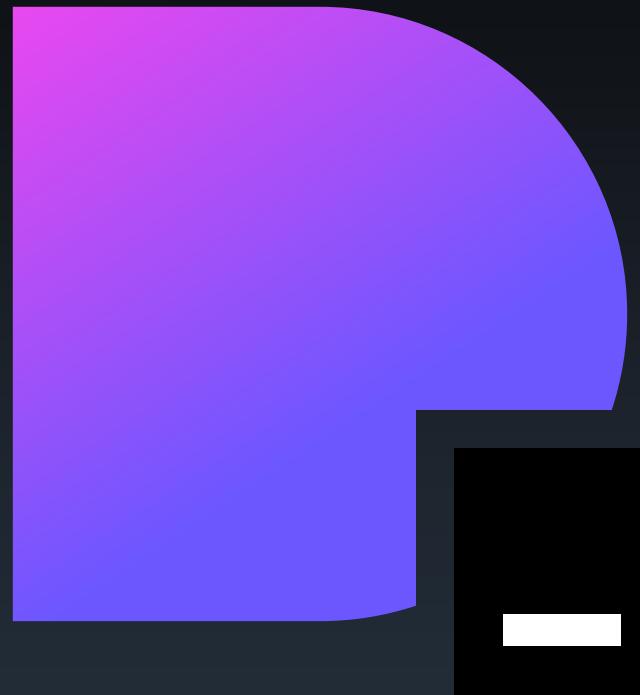
Паша Финкельштейн, JetBrains

Михаил Марюфич, [OK.ru](#)

О себе

- 14 лет в ИТ
- 11 в разработке с разных концов
- 2 года в DE
- Из них год в Big Data Tools *

* *Big Data Tools – инструмент для упрощения жизни дата инженеров*



Михаил Марюфич

- Закончил Мат-Мех СПбГУ и Computer Science Center
- Machine Learning Engineer в [OK.ru](#)
- Делаю МЛ штуки и внедряю их в продакшн
- Увлекаюсь воспроизведимостью



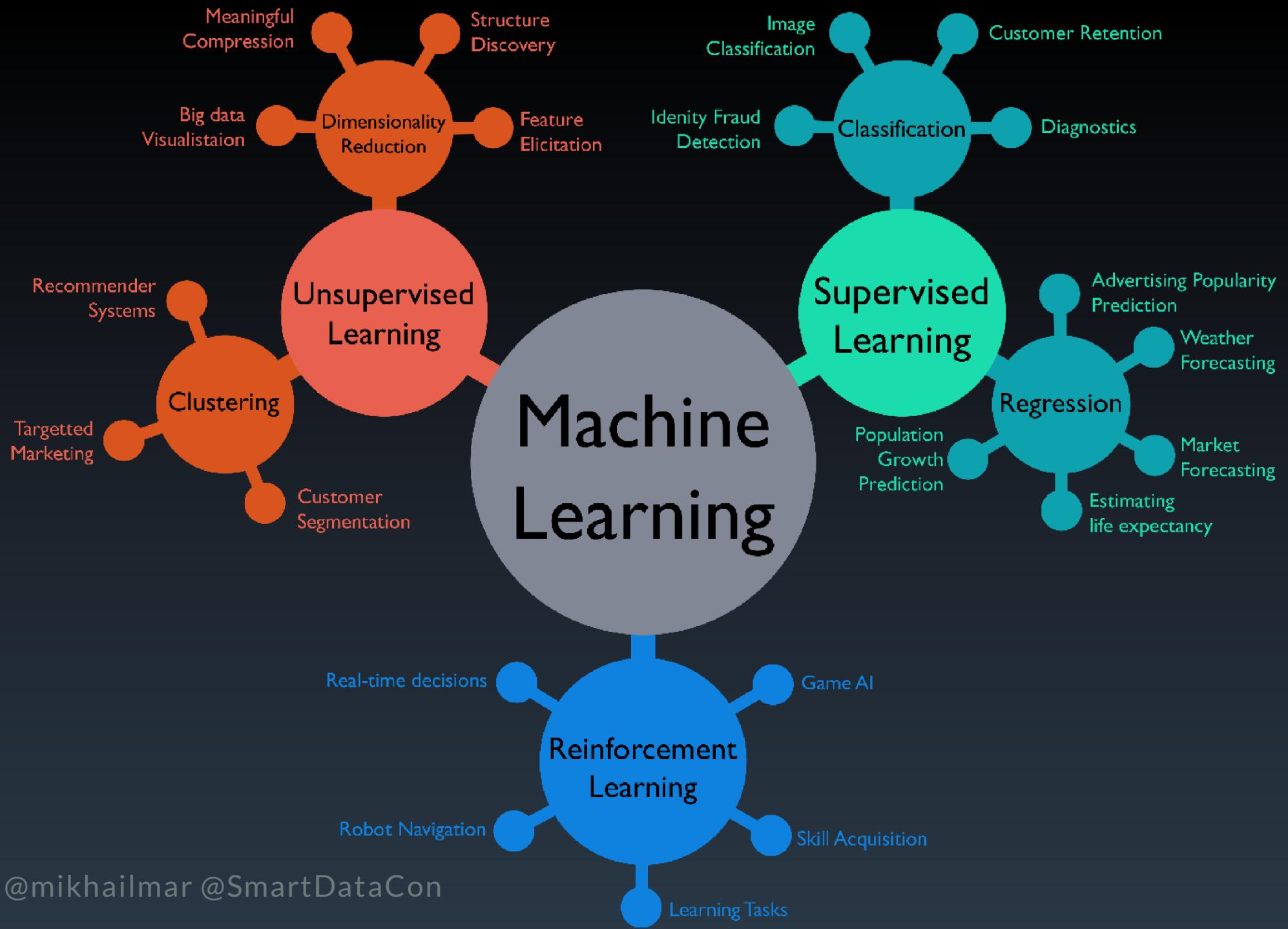
Это доклад о том, что такое MLOps, зачем он нужен и как можно попробовать его делать

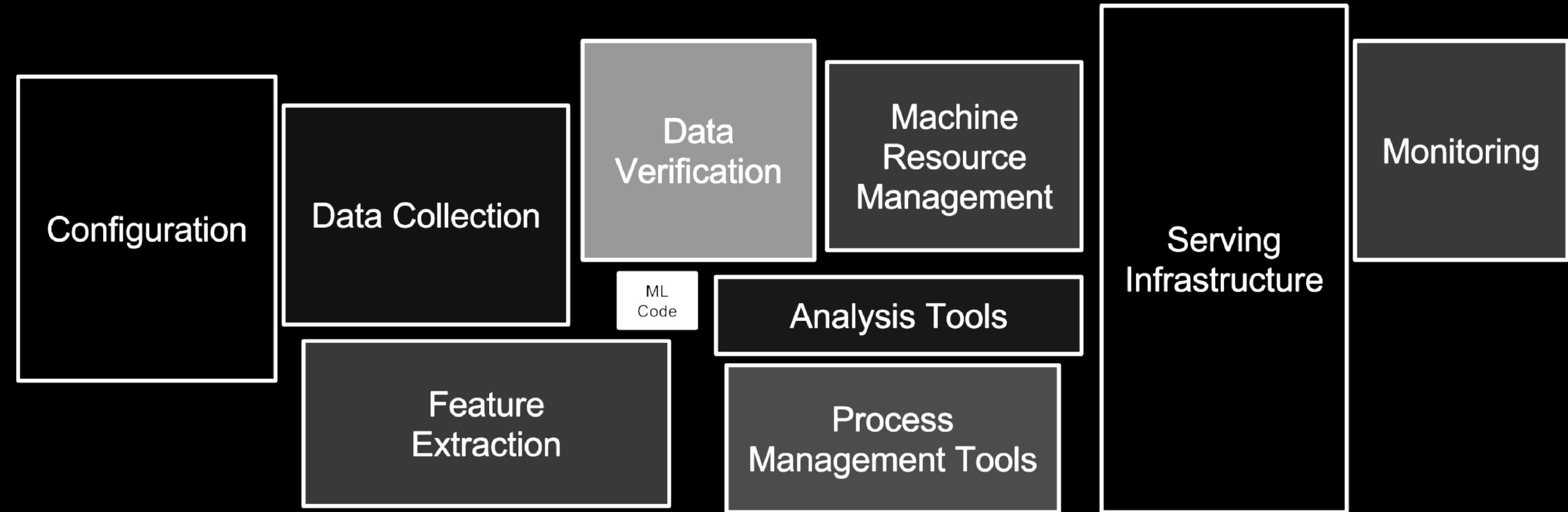
Концепции общие, инструменты меняются

О древнейшей истории

- Буча началась с гиганта
- И гигантом был Google

<https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf>

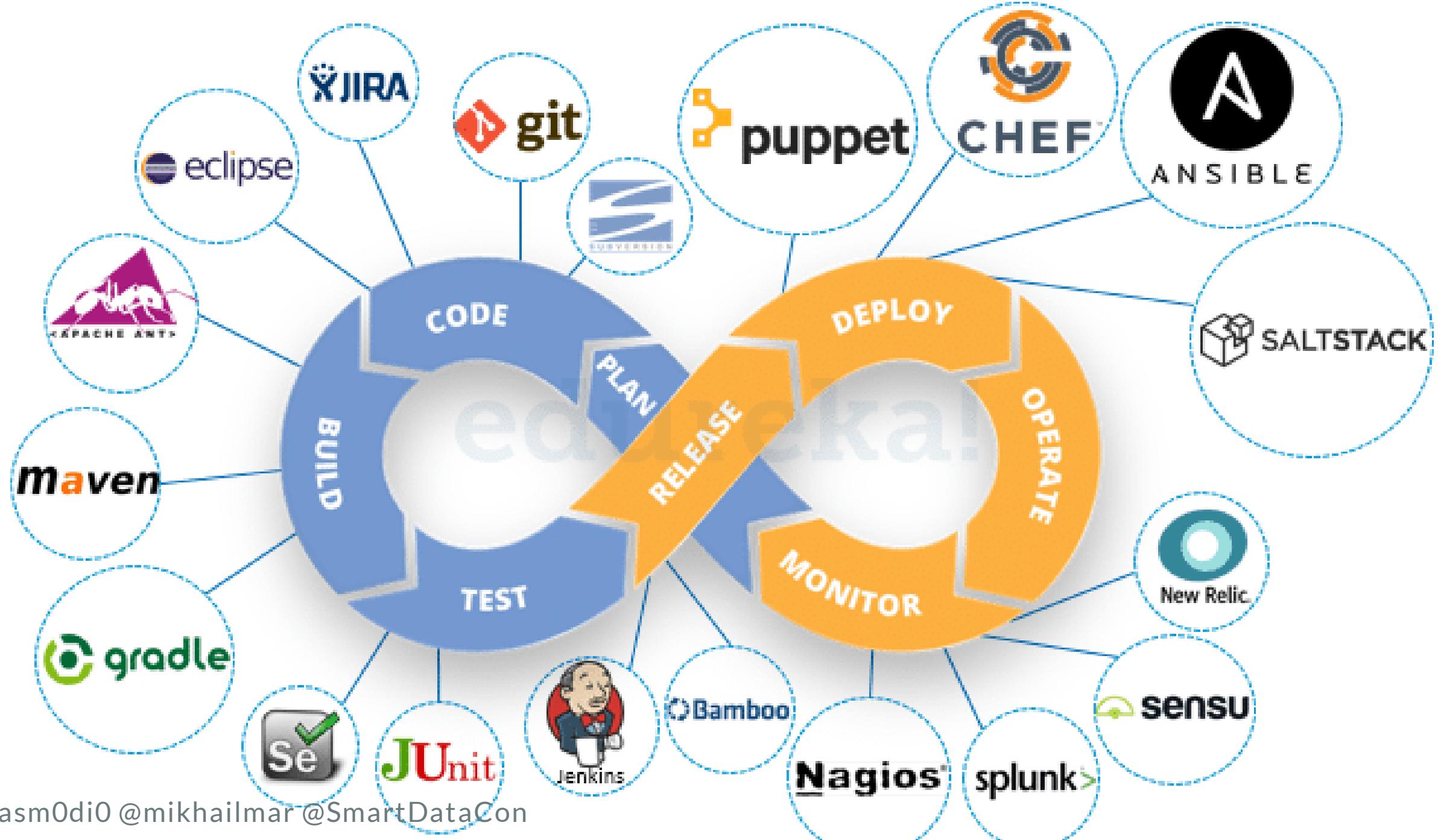




“ Мы все неправы! ”

Google

Потому что не продуктивизируем свой ML код.



DevOps для обычной разработки

Тем временем в обычной разработке

- Инструментов тонна
- Все знают, как продуктизировать разработку
- Все понимают, где светлое будущее

И конечно же вокруг этого навёрнута тонна практик

Operations Anti-Patterns, DevOps Solutions

Jeffery D. Smith



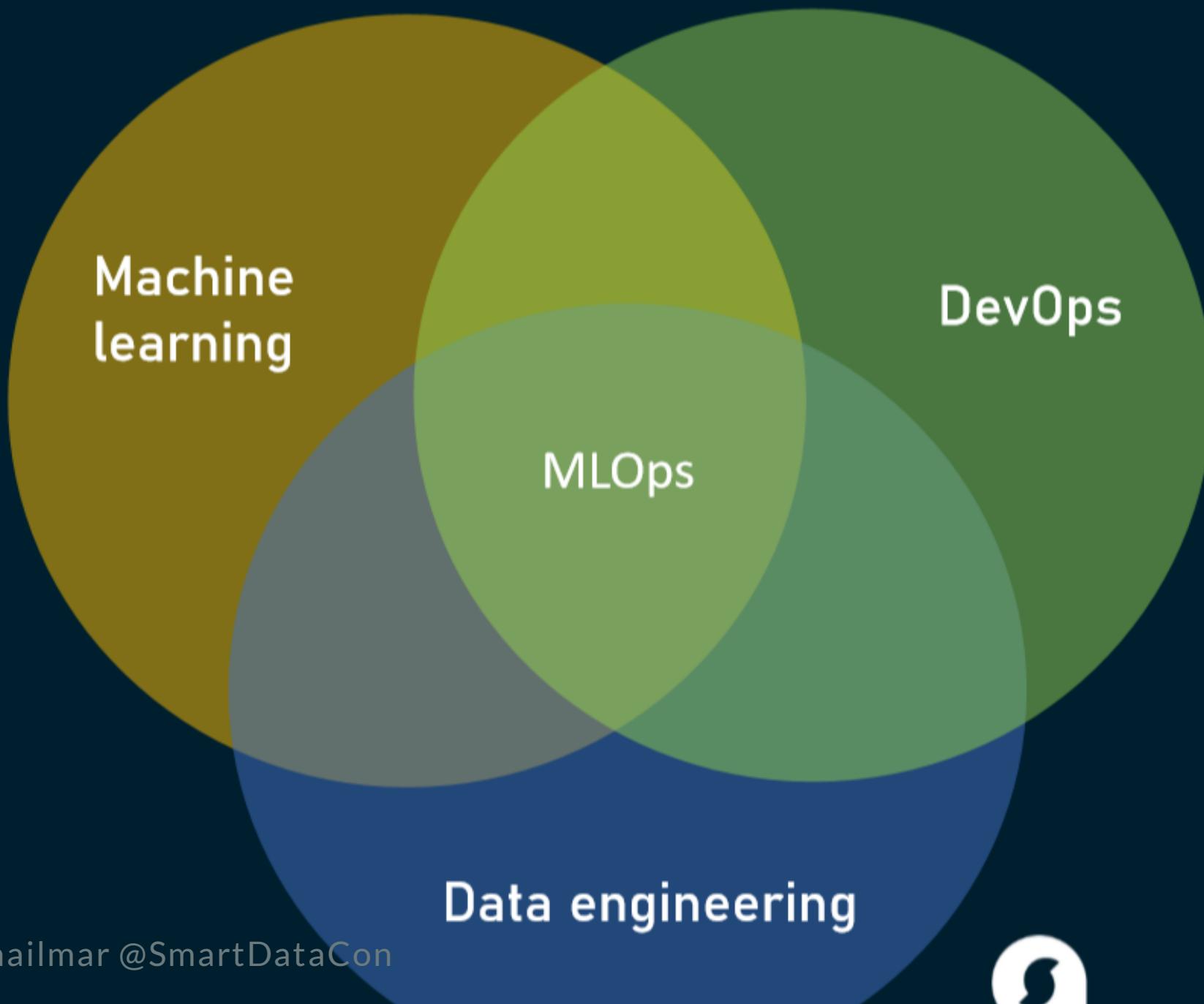
“ DevOps isn’t about tools, but
about how teams work
together ”

**Новая книга про
анти-паттерны Ops и
решения DevOps**

@asm0di0 @mikhailmar @SmartDataCon

Так что, MLOps – это просто DevOps?

Нет!



**MLOps – это ещё одна ступень
специализации**

Как устроен ML

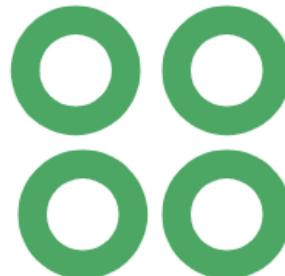


Data

Schema

Sampling over Time

Volume



Model

Algorithms

More Training

Experiments



Code

Business Needs

Bug Fixes

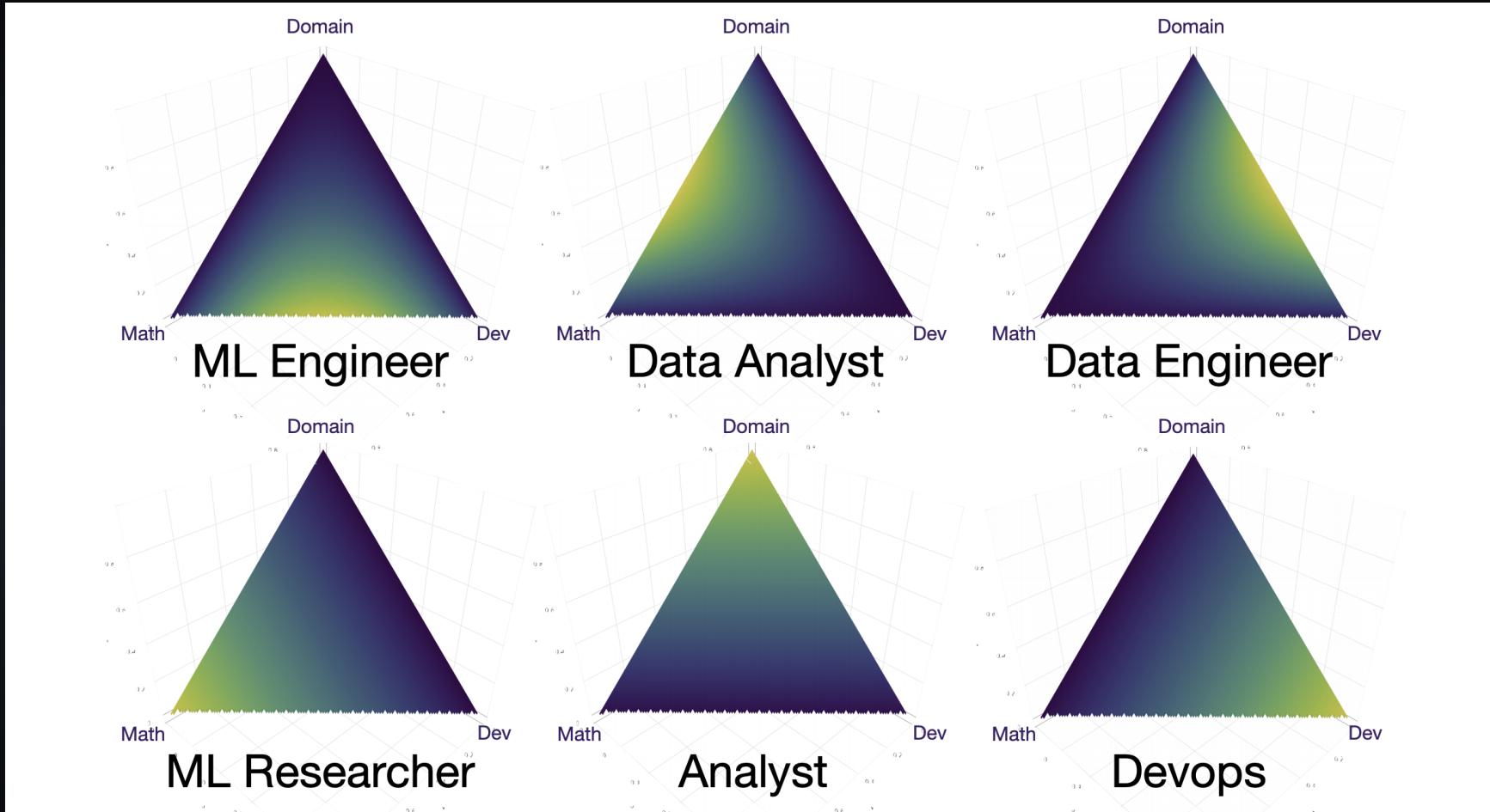
Configuration



А кто такие дата саентисты?

- Data – данные, тут всё понятно.
 - копаются в реке и находят золото
- Scientists – учёные. Экспериментаторы

Роли в ML разработке



Требования к эксперименту

- Формулирование гипотезы
- Описание специфических условий
- Воспроизводимость
- Протоколирование

“ MLOps – это способ
сделать эксперименты
научным, а не наколеночным ”

Эйнштейн



Я всегда верю цитатам в интернете

IN EASTERLY
AS IN THE HEARTS OF THE PEOPLE
FOR WHOM HE SAVED THE UNION
THE MEMORY OF ABRAHAM LINCOLN
IS ENSHRINED FOREVER

Курт Кобейн



Как устроен ML



Data

Schema

Sampling over Time

Volume



Model

Algorithms

More Training

Experiments



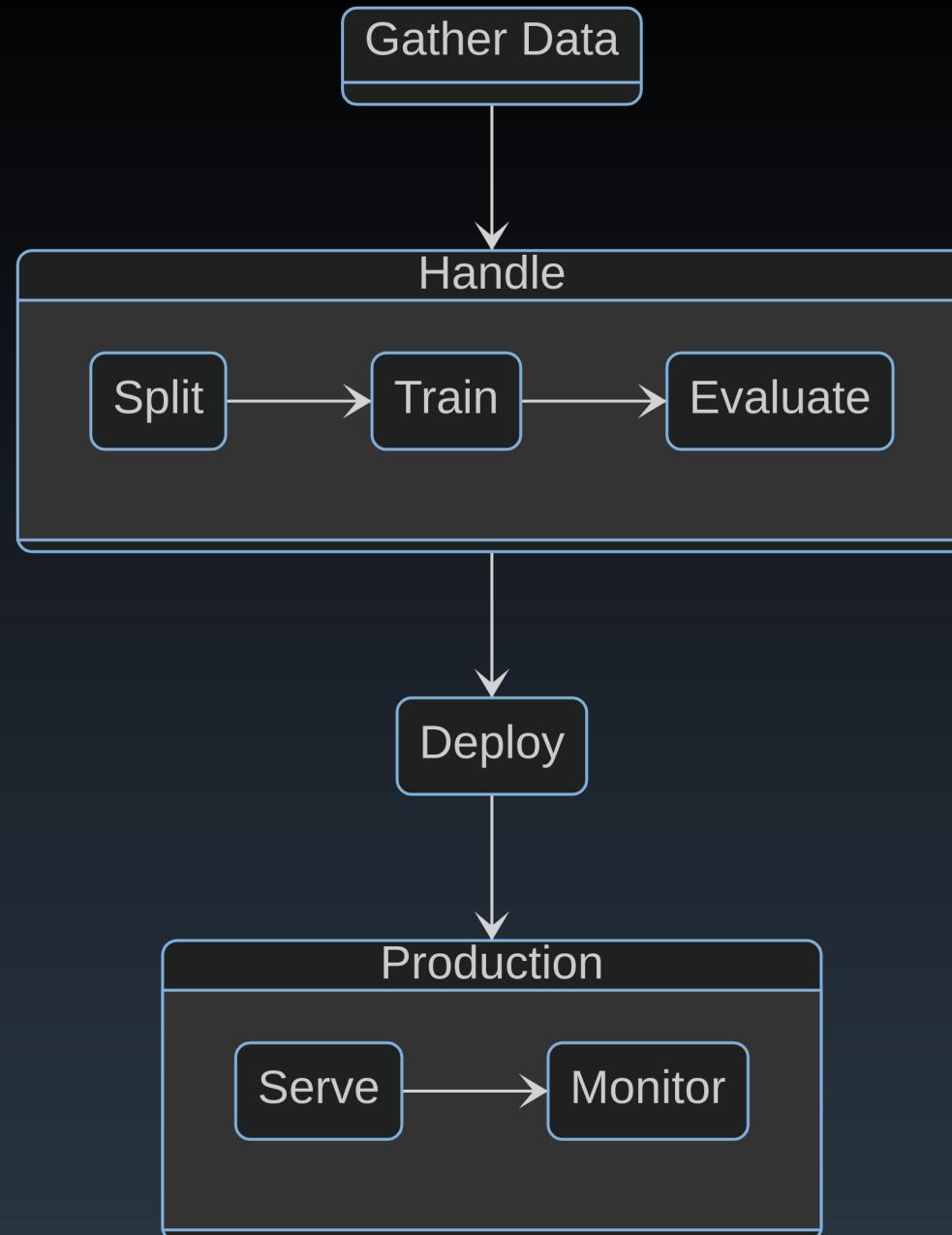
Code

Business Needs

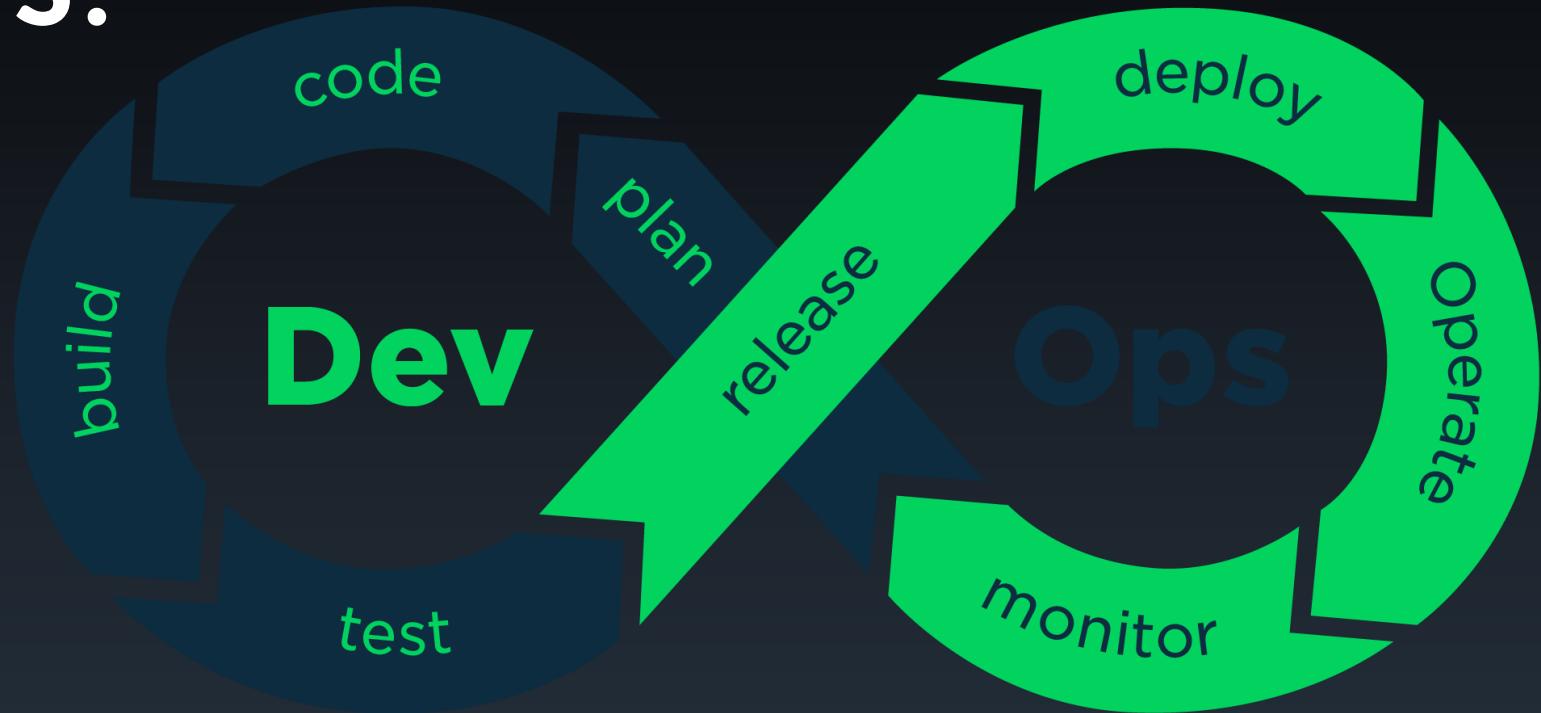
Bug Fixes

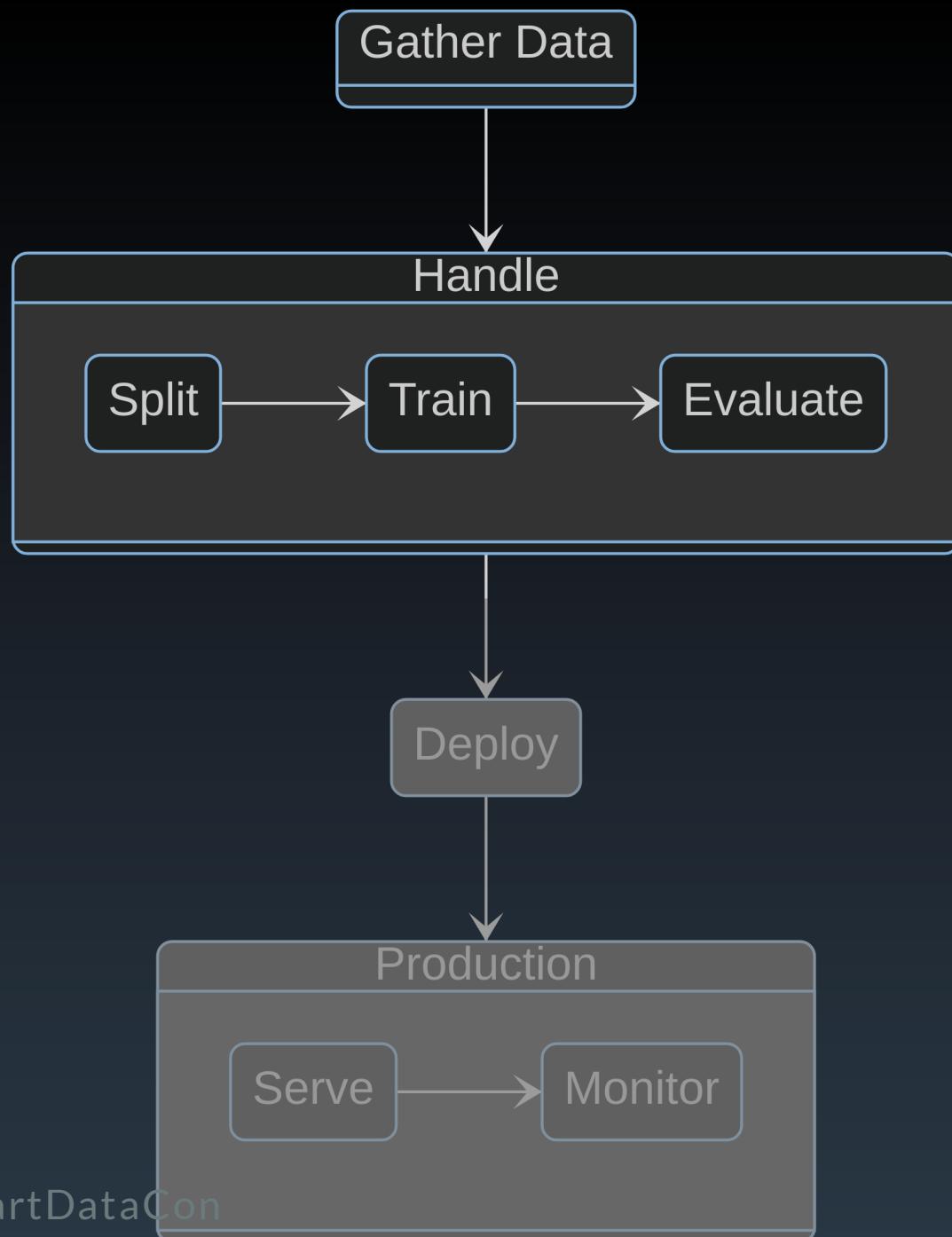
Configuration

ML workflow



А ЧЕМ ПРИНЦИПИАЛЬНО отличается от DEVOPS?





Gather Data

Это же билд!

Handle

Split

Train

Evaluate

Deploy

Production

Serve

Monitor

Gather Data

Это же билд!

Handle

Split

Train

Evaluate

Deploy

Production

Serve

Monitor

PS: только очень сложный

Что тут пошло не так?

- Всё одним куском
- Протокола эксперимента нет
- Информации об исходных данных нет
- Результаты меняются от запуска к запуску

Что делать?

1



download_data.py



Data

2



splitter.py



3



evaluation.py



decision_tree.py



model.pkl



metrics.json

Разве это не очевидно?

НЕТ!

```
In [21]: import os  
import platform
```

Jupyter

```
In [22]: platform
```

```
Out[22]: <module 'platform' from '/opt/anaconda3/lib/python3.6/platform.py'>
```

```
In [23]: from pyspark import SparkConf  
from pyspark import SparkContext
```

```
In [24]: sc = SparkContext.getOrCreate()
```

```
In [25]: sc.getConf().getAll()
```

```
Out[25]: [('spark.app.id', 'local-1536486752230'),  
          ('spark.driver.port', '42132'),  
          ('spark.rdd.compress', 'True'),  
          ('spark.serializer.objectStreamReset', '100'),  
          ('spark.master', 'local[*]'),  
          ('spark.executor.id', 'driver'),  
          ('spark.submit.deployMode', 'client'),  
          ('spark.ui.showConsoleProgress', 'true'),  
          ('spark.app.name', 'pyspark-shell'),  
          ('spark.driver.host', 'spark-master.c.mozn-location.internal')]
```

```
In [26]: import numpy as np  
TOTAL = 1000000  
dots = sc.parallelize([2.0 * np.random.random(2) - 1.0 for i in range(TOTAL)]).cache()  
print("Number of random points:", dots.count())  
stats = dots.stats()  
print('Mean:', stats.mean())  
print('stdev:', stats.stdev())
```

```
Number of random points: 1000000
```

```
Mean: [0.00075275 0.00051083]
```

```
stdev: [0.57730526 0.57748448]
```

```
In [21]: import os  
import platform
```

Jupyter

```
In [22]: platform  
Out[22]: <module 'platform' from '/opt/anaconda3/lib/python3.6/platform.py'>
```

```
In [23]: from pyspark import SparkConf  
from pyspark import SparkContext
```

- Самый популярный (и достаточно удобный) инструмент Data Scientist'a

```
In [24]: sc = SparkContext.getOrCreate()
```

```
In [25]: sc.getAll()
```

- Код в перемешку с Markdown
- Написан и исполняется в непредсказуемом порядке

```
Out[25]: [('spark.app.id', 'local-1516486752130'),  
          ('spark.driver.host', '127.0.0.1'),  
          ('spark.rdd.compress', 'True'),  
          ('spark.serializer.objectStreamReset', '100'),  
          ('spark.reducer.maxPartitions', '1'),  
          ('spark.executor.id', 'driver'),  
          ('spark.submit.deployMode', 'client'),  
          ('spark.ui.showConsoleProgress', 'true'),  
          ('spark.app.name', 'pyspark-shell'),  
          ('spark.driver.host', 'spark-master.c.mozn-location.internal')]
```

```
In [26]: import numpy as np  
TOTAL = 1000000  
dots = sc.parallelize([2.0 * np.random.random(2) - 1.0 for i in range(TOTAL)]).cache()  
print("Number of random points:", dots.count())  
stats = dots.stats()  
print('Mean:', stats.mean())  
print('stdev:', stats.stdev())
```

```
Number of random points: 1000000  
Mean: [0.0065275 0.00051082]  
stdev: [0.57730526 0.57748448]
```

Придётся переучивать



DVC

- Создавался специально в пару к гиту
- Изначально был похож на LFS (git-lfs)
- Оброс функциональностью

Примеры

```
git add  
dvc add
```

Добавить какой-то файл под контроль git/dvc

Примеры

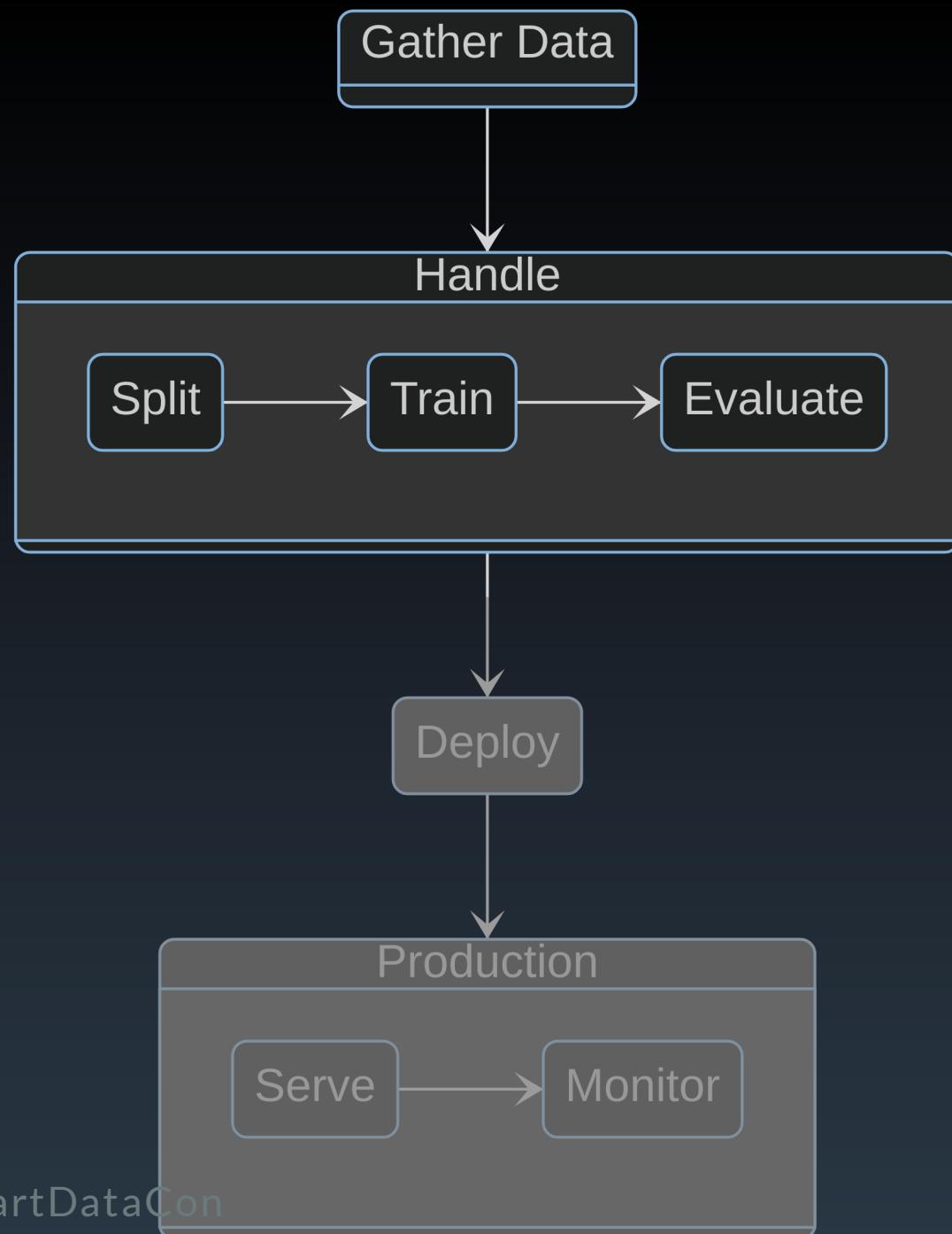
```
git commit  
dvc commit
```

Сохранить состояние файла

Примеры

```
git diff  
dvc diff
```

Посмотреть разницу с каким-то состоянием





The talk is cheap

Show us the demo!

ОК, НО ЭТО КОНСОЛЬ

- Молодёжь 🤓 не любит терминалы
- Этим неудобно пользоваться при командной разработке
- Тимлиду может быть неудобно работать с метриками



[Overview](#) [Diff](#) [Commits](#)

PR - это очень удобно

Details



Neural Network Trainer Service created a pull request 9 hours ago

auto request [XPRM-16333] -> [master]

Activity



What do you want to say?



Mikhail Maryufich MERGED XPRM-16333 to master in commit [5175918ed7c](#) 8 hours ago



Mikhail Maryufich marked the pull request as APPROVED 8 hours ago

myflow™

Easy



```
mlflow.sklearn.autolog()  
remote_server_uri = "http://127.0.0.1:5000"  
mlflow.set_tracking_uri(remote_server_uri)  
mlflow.set_experiment("udemy-courses")
```

MLFlow Tracking

The screenshot shows the MLflow Tracking interface. At the top, there's a navigation bar with links for 'Experiments' (which is active), 'Models', 'GitHub', and 'Docs'. Below the navigation, the page title is 'udemy-courses'. On the left, there's a sidebar titled 'Experiments' with a search bar and two entries: 'Default' and 'udemy-courses'. The 'udemy-courses' entry is selected and highlighted in light blue. To the right of the sidebar, the main content area displays the experiment details: 'Experiment ID: 1' and 'Artifact Location: ./mlruns/1'. Below this, there's a section for 'Notes' which says 'None'. Underneath, there's a search bar with the query 'metrics.rmse < 1 and params.model = "tree" and tags.mlflow.source.type =', a 'State' dropdown set to 'Active', and 'Search' and 'Clear' buttons. The main table below shows 'Showing 2 matching runs'. The table has columns for Start Time, Run Name, User, Source, Version, bootstrap, ccp_alpha, copy_X, training_r, and various estimator metrics. Two rows of data are listed, both associated with the user 'finkel' and the source 'train_r'. The first row has a checked checkbox in the Start Time column and a checked 'True' in the copy_X column. The second row also has a checked checkbox in the Start Time column and a checked 'True' in the copy_X column.

	Start Time	Run Name	User	Source	Version	bootstrap	ccp_alpha	copy_X	training_r	training_r	training_r	estimator	estimator
<input checked="" type="checkbox"/>	2020-11-23 19:1 -		finkel	train_r	9f9980	True	0.0	-	1236.1	9678...	0.897	skle...	Rand...
<input checked="" type="checkbox"/>	2020-11-23 19:1 -		finkel	train_r	479713	-	-	True	3799.9	8854...	0.062	skle...	Linea...

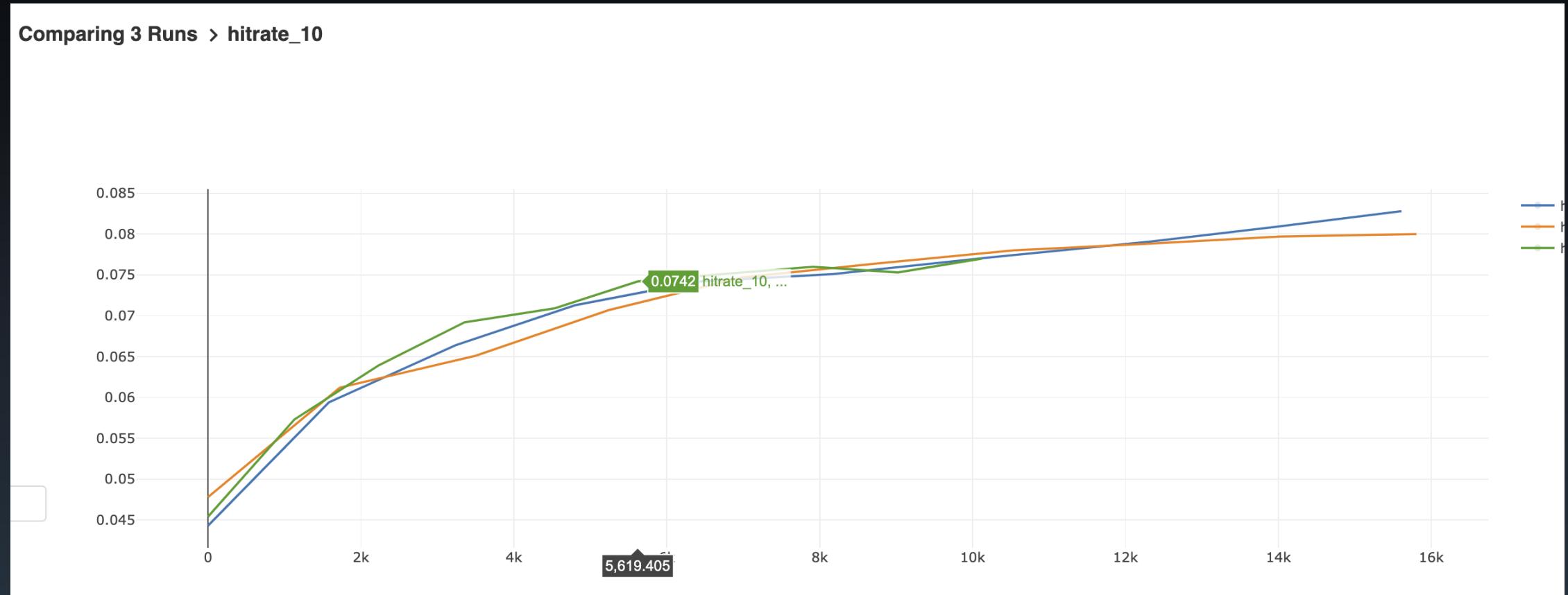
@asm0di0 @mikhailmar @SmartDataCon

MLFlow Tracking - сравнение запусков

Grid search demo > Comparing 4 Runs

Run ID:	84e2712f7a864c509a52b255394d5e5a	2ff7d4c31f7746b6a18af504e6430014	a8c0994d0d4147e2824e6109d93d2cd5	99ce9f6dad914c3d9a7e744979dd930c
Run Name:	Fork for configuration=42	Fork for configuration=41	Fork for configuration=40	Fork for configuration=39
Start Time:	2019-11-01 17:51:38	2019-11-01 17:51:02	2019-11-01 17:51:01	2019-11-01 17:50:25
Parameters				
elasticNetParam	0.5924460687674582	0.0924460687674582	0.9049460687674582	0.4049460687674582
regParam	0.07031306629069149	0.020313066290691496	0.014063066290691496	0.0640630662906915
Metrics				
auc on test ↗	0.74	0.744	0.744	0.742
auc on train ↗	0.736	0.739	0.738	0.736
auc_pr on test ↗	0.396	0.394	0.398	0.394
auc_pr on train ↗	0.398	0.398	0.401	0.396

MLFlow Tracking - сравнение запусков



Registered Models

[Create Model](#)

Filter registered models by name or t...



Name	Latest version	Aliased versions	Created by	Last modified	Tags
iris_model_dev	Version 17			2023-09-25 12:50:...	—
iris_model_prod	Version 11	@ champion: Version 11 +3		2023-09-25 16:11:...	—
iris_model_staging	Version 11			2023-09-25 12:46:...	—
iris_model_testing	Version 1			2023-09-27 13:17:...	—
mnist_model_dev	Version 12			2023-09-25 12:39:...	—
mnist_model_prod	Version 8	@ challenger: Version 8 +1		2023-09-26 23:41:...	—
mnist_model_staging	Version 8			2023-09-25 12:51:...	—

New model registry UI [◀ Previous](#) [Next ▶](#)

25 / page

Default >

magificent-seal-662

Run ID: f34c3372e3bf4f278cb3fa73cd0815ab

Date: 2023-09-25 14:05:17

Source: ipython

User: jerry.liang

Duration: 2.5s

Status: FINISHED

Lifecycle Stage: active

> Description [Edit](#)

> Parameters (2)

> Metrics

> Tags

▼ Artifacts

sk_models

- MLmodel
- conda.yaml
- input_example.json
- model.pkl
- python_env.yaml
- requirements.txt

Full Path

Register Model

Model

[+ Create New Model](#)

Model Name

iris_model_testing

Cancel

Register

MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. You can also [register it to the model registry](#) to version control

Model schema

Input and output schema for your model. [Learn more](#)

Make Predictions

Predict on a Spark DataFrame:

```
import mlflow
from pyspark.sql.functions import struct, col
logged_model = 'runs:/f34c3372e3bf4f278cb3fd73cd0815ab/sk_models'
```

Как модель там окажется?

```
from mlflow.tracking import MlflowClient  
  
client = MlflowClient()  
client.create_registered_model("sk-learn-random-forest-reg-model")
```

```
client = MlflowClient()  
result = client.create_model_version(  
    name="sk-learn-random-forest-reg-model",  
    source="mlruns/0/d16076a3ec534311817565e6527539c0/artifacts/sklearn-model",  
    run_id="d16076a3ec534311817565e6527539c0"  
)
```

Что дальше?

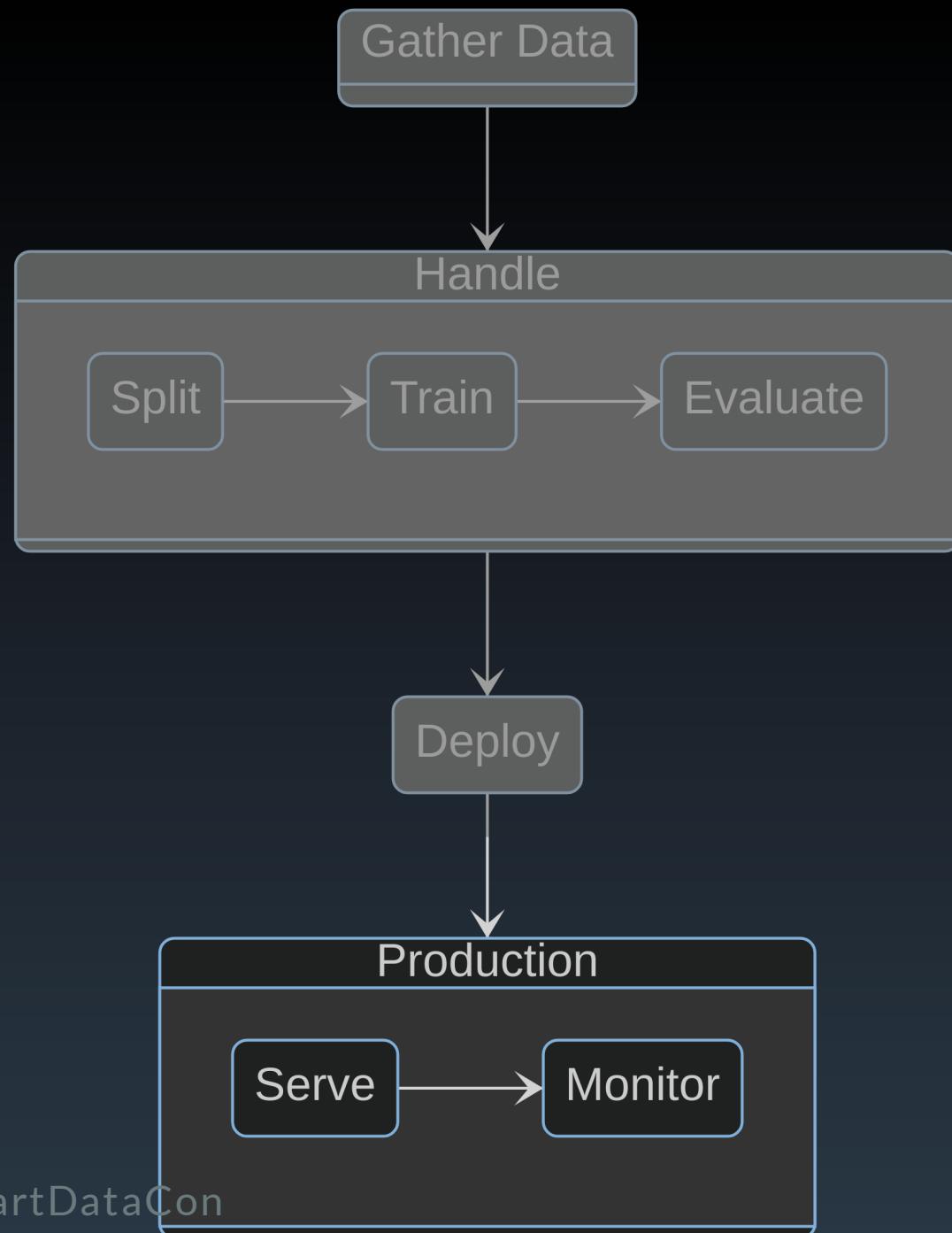
- У нас есть модельки
- У нас есть каталог моделек
- Тестируем!

Тестируем

Мы уже знаем всё, что надо знать про модельки:

1. Среднеквадратичное отклонение
2. Коэффициент детерминации
3. Метрики, специфичные для нашей модели

Всё это можно делать в целом в любом CI!

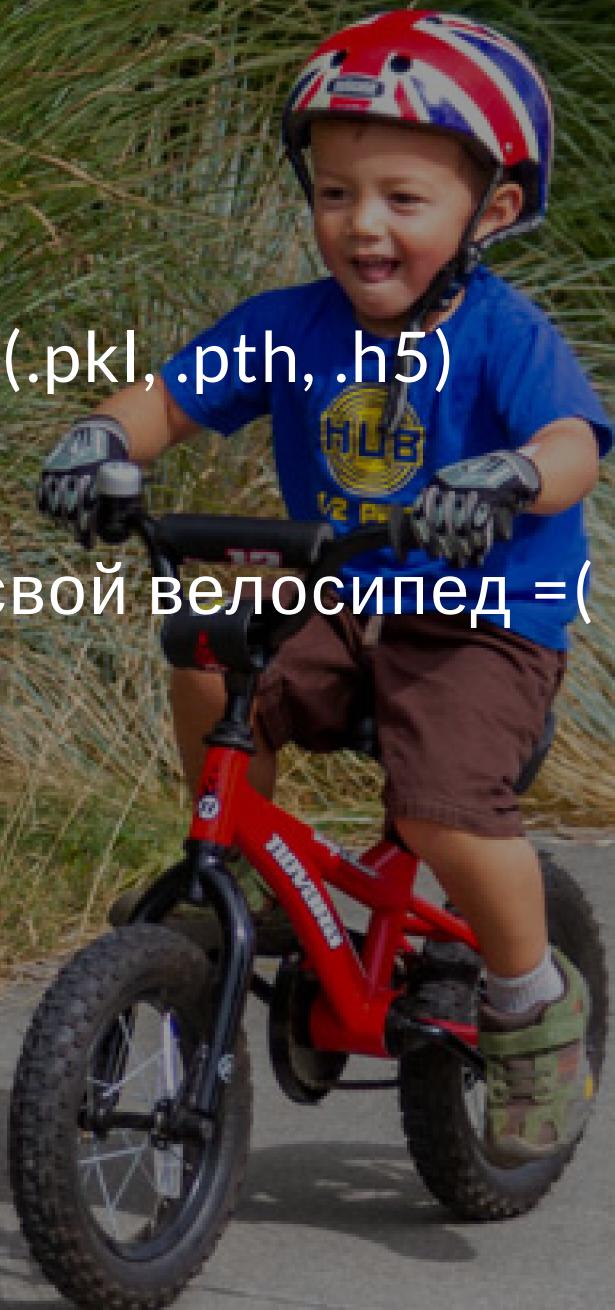


Serve

- Есть ML MODEL - это бинарник (.pkl, .pth, .h5)
- Нужно делать к ней запросы

Serve

- Есть ML MODEL - это бинарник (.pkl, .pth, .h5)
- Нужно делать к ней запросы
- И каждый стремится написать свой велосипед =(



MLFLOW SERVING



```
$ mlflow models serve -m runs:/my-run-id/model-path &  
  
$ curl http://1127.0.0.1:5000/invocations -H 'Content-type: application/json' -d '{  
    "columns": ["a", "b", "c", "d"],  
    "data": [[1, 2, 3], [4, 5, 6]]  
}'
```



1. Containerise
2. Deploy
3. Monitor

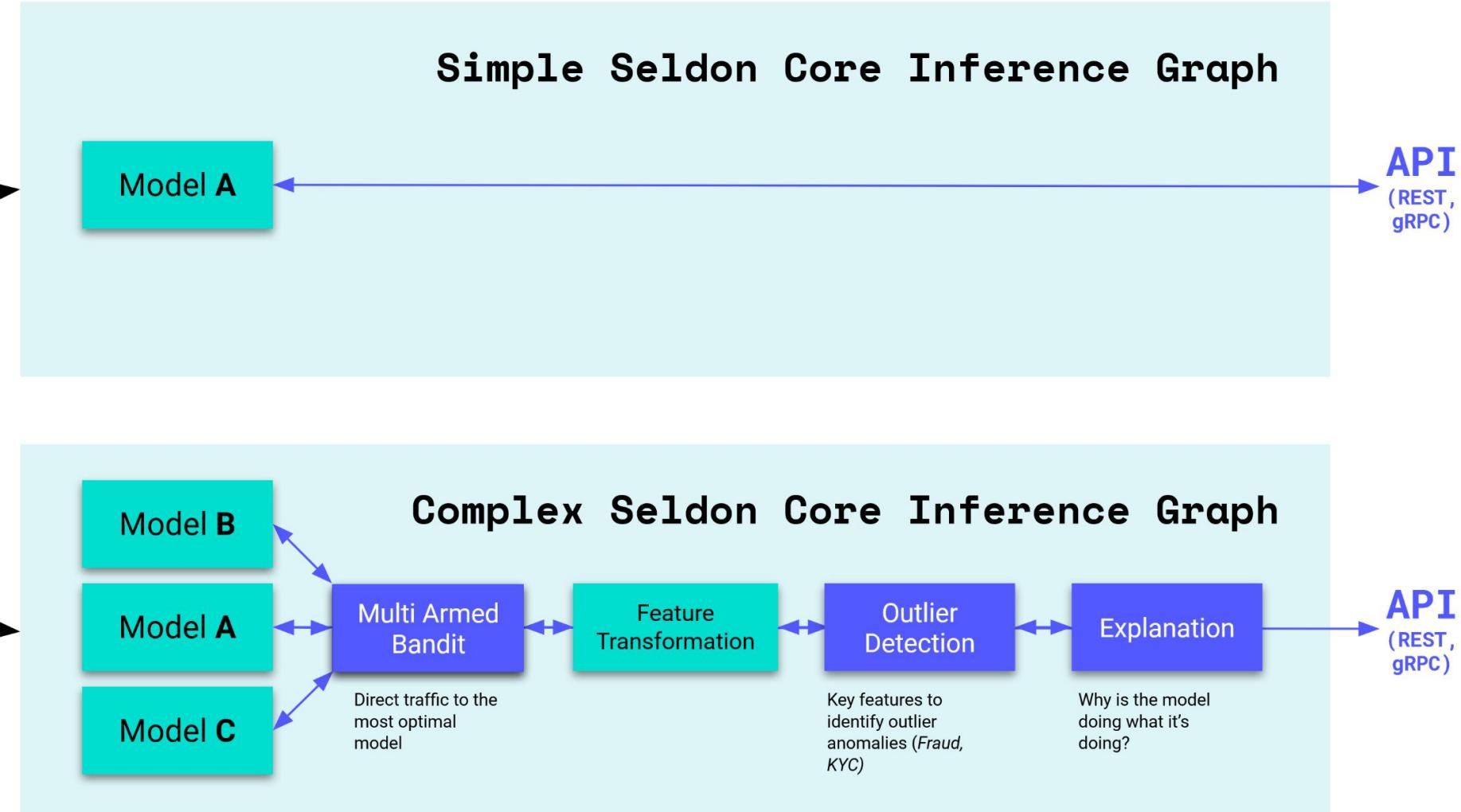
From model binary



Or language wrapper



Into fully fledged microservice





Show us the demo!

Мониторинг

2 типа инструментов:

1. ML-Специфичные
2. Общие

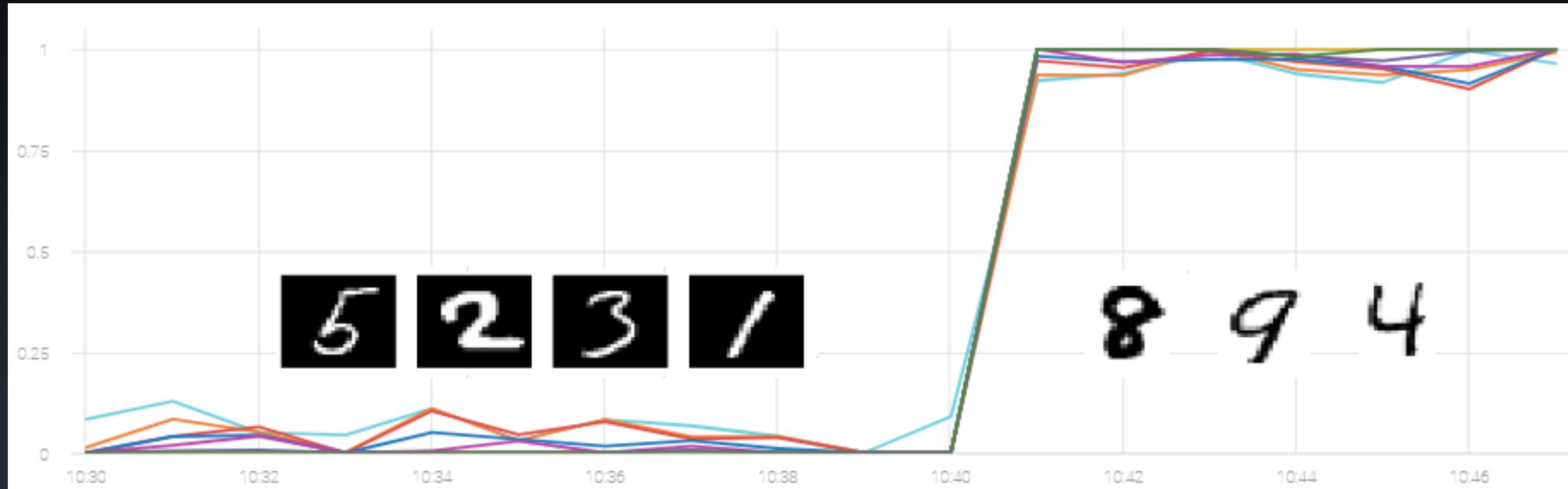
MLWatcher

- range, mean, std, median, q25, q50, q75, iqr for any continuous values (probabilities, features)
- count, frequency for any discrete values (labels, classes)

Python-агент, который позволяет мониторить *статистические метрики*

Sampling

Пример



Метрики улетают когда цвета внезапно инвертируются

Использование

1. Уложить MLWatcher рядом со своим приложением 
2. Поднять сервер
3. Начать с определённой частотой репортировать метрики на сервер

Общие метрики

Мониторить параметры модели - мало

Надо мониторить

1. Системный метрики

1.1. la

1.2. free

1.3. latency...

2. Бизнес-метрики модели (например потерянные клиенты)

Инструменты

- Prometheus
- Grafana
- Zabbix etc

All predictor All version All model_name All model_image All model_version All

Seldon Core API Dashboard

Global Request Rate

143 ops

Success

100%

4xxs

N/A

5xxs

0 ops

Models

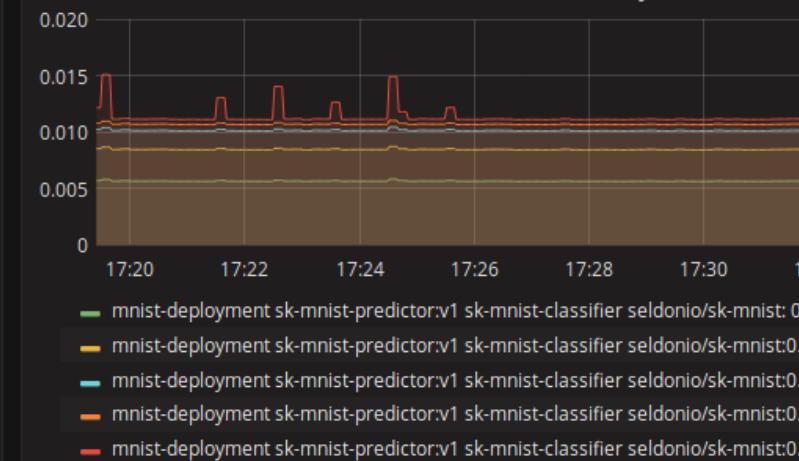
Reqs/sec to seldonio/sk-mnist



seldonio/sk-mnist Reward



seldonio/sk-mnist Latency



Мониторинг – важнейшая часть продуктивизированного ML

Без него не имеет смысла вообще продуктивизировать – будет непредсказуемо

Чему мы научились

1. MLOps – это процессы и командное взаимодействие
2. Чтобы построить MLOps надо иметь базовый набор инструментов
3. Кроме стандартных инструментов есть и непривычные: DVC, MLFlow
4. Продуктивизация ML – это большой путь, который надо пройти

QA

Паша Финкельштейн: if ( @asm0di0 else @asm0dey

Михаил Марюфич: @mikhailmar