Twitter Thread by Goku Mohandas

Goku Mohandas
@GokuMohandas

All the @madewithml machine learning fundamentals & MLOps lessons are released!

- ■ Project-based
- ■ Intuition & application (code)
- ■ 26K+ GitHub ■■
- ❤■ 30K+ community
- ■ 47 lessons, 100% open-source

https://t.co/XIhD3wl1DA

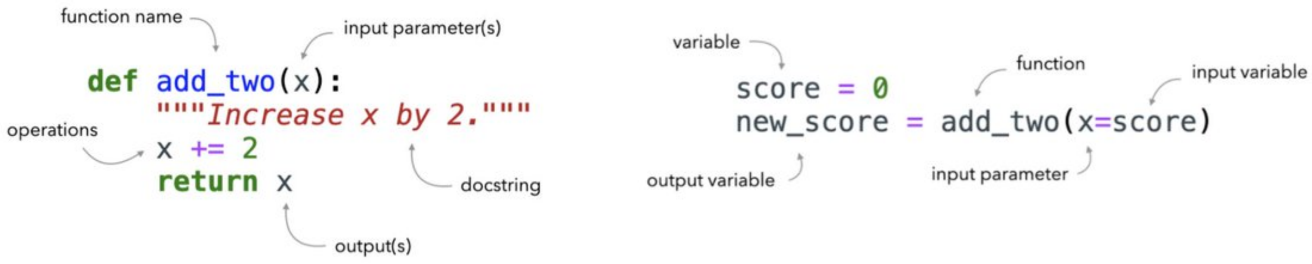
■ Thread on details & lesson highlights ■

Who is this course for?

- ■ Software engineers / Data scientists looking to learn how to responsibly create ML systems.
- ■ College grads looking to learn the practical skills they'll need for the industry.
- ■ Product Managers who want to develop a technical foundation.

We start with lessons on the fundamentals of ML through intuitive explanations, clean code and visualizations.

■ Foundations
- Python (variables, functions, classes, decorators)
- NumPy (numerical analysis)
- Pandas (data analysis)
- PyTorch (operations, gradients)




```python
# Define the function
def add_two(x):
    """Increase x by 2."""
    x += 2
    return x
```

```python
# Use the function
score = 0
new_score = add_two(x=score)
print (new_score)
```

```
2
```

Then we dive into implementing basic ML algorithms 1■ from scratch then 2■ in PyTorch. Starting from simple models → complex models.

■ Modeling
- Linear Regression
- Logistic Regression
- Neural Networks
- Data Quality (■■ very important)
- Utilities (for loading and training)

`Step 4`: Calculate the gradient of loss $J(\theta)$ w.r.t to the model weights.

$$J(\theta) = \frac{1}{N}\sum_i (y_i - \hat{y}_i)^2 = \frac{1}{N}\sum_i (y_i - X_iW)^2$$

$$\rightarrow \frac{\partial J}{\partial W} = -\frac{2}{N}\sum_i (y_i - X_iW)X_i = -\frac{2}{N}\sum_i (y_i - \hat{y}_i)X_i$$

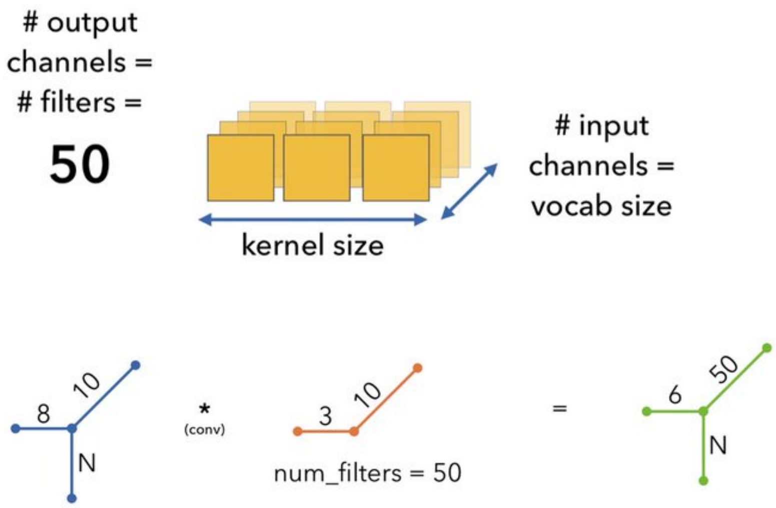
$$\rightarrow \frac{\partial J}{\partial b} = -\frac{2}{N}\sum_i (y_i - X_iW)1 = -\frac{2}{N}\sum_i (y_i - \hat{y}_i)1$$

```python
# Backpropagation
dW = -(2/N) * np.sum((y_train - y_pred) * X_train)
db = -(2/N) * np.sum((y_train - y_pred) * 1)
```

We wrap up the fundamentals by implementing deep learning algorithms in PyTorch.

■ Deep Learning
- CNNs
- Embeddings
- RNNs
- Transformers

■ We motivate the need for specific architectures and additional complexity as we implement each method.



$$W_1 = \frac{W_2 - F + 2P}{S} + 1 = \frac{8 - 3 + 2(0)}{1} + 1 = 6$$

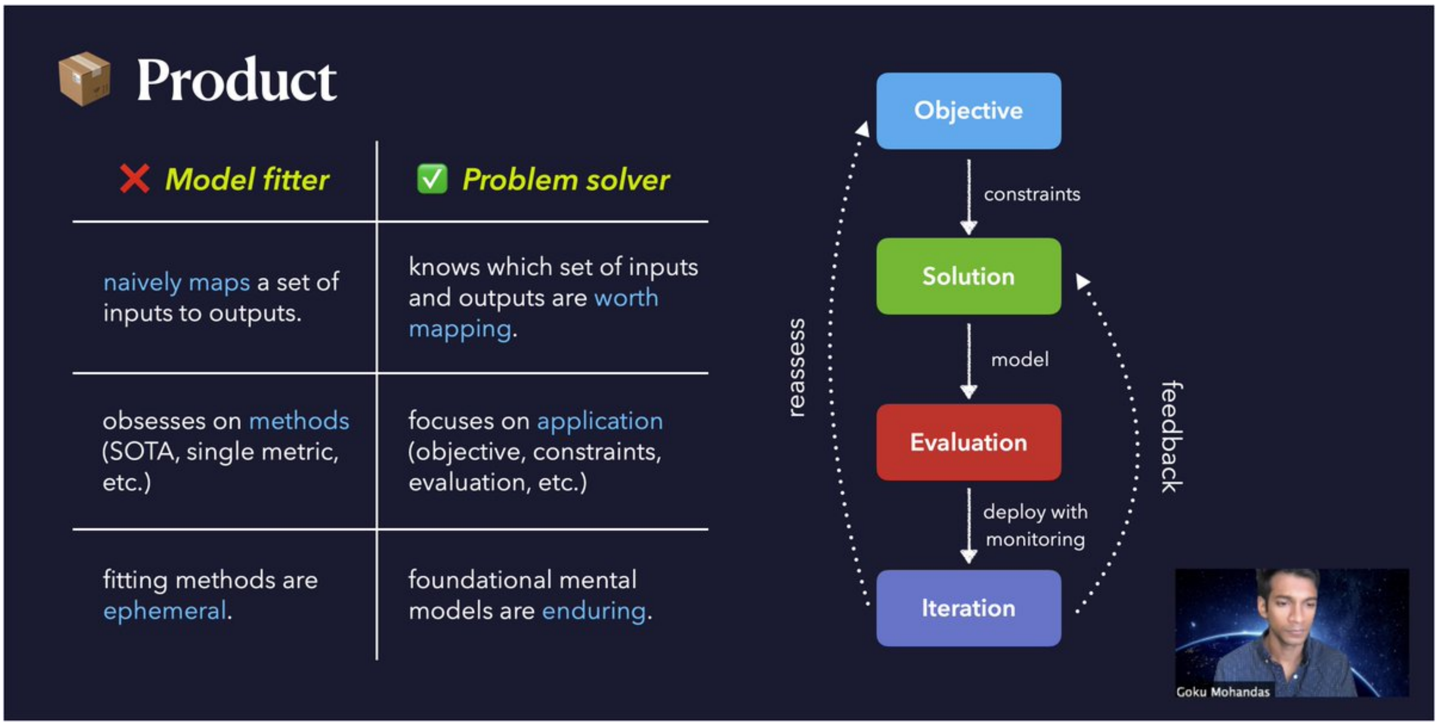
$$H_1 = \frac{H_2 - F + 2P}{S} + 1 = \frac{1 - 1 + 2(0)}{1} + 1 = 1$$

$$D_2 = D_1$$

Variable	Description
W	width of each input = 8
H	height of each input = 1
D	depth (# of channels)
F	filter size = 3
P	padding = 0
S	stride = 1

The first MLOps lessons are on the Product development and iteration cycle.

■ Product
- Identify the core objective.
- Design a solution with constraints.
- Evaluation strategies that avoid bias.
- Iterate via feedback and motivate adding complexity.



Next we dive into exploring and transforming our data.

■ Data
- Labeling (data worth modeling, active learning)
- Preprocessing (prepare + transform)
- Exploration (answering questions)
- Splitting (multi-label classification)
- Augmentation (nlpaug, transformation functions)

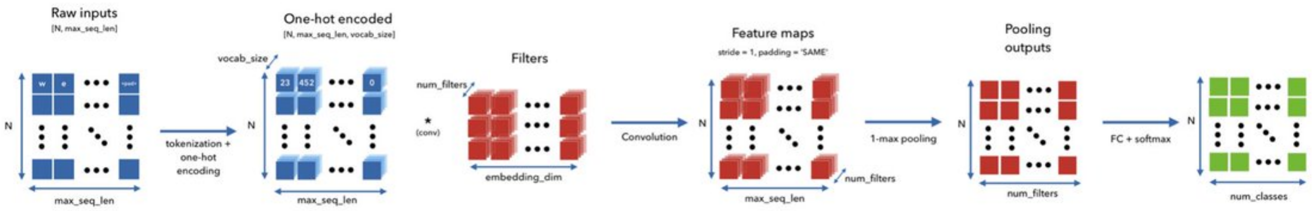
```python
# Load tags
url = "https://raw.githubusercontent.com/GokuMohandas/MadeWithML/main/datasets
tags = json.loads(urlopen(url).read())
tags_dict = {}
for item in tags:
    key = item.pop("tag")
    tags_dict[key] = item
print (f"{len(tags_dict)} tags")
```

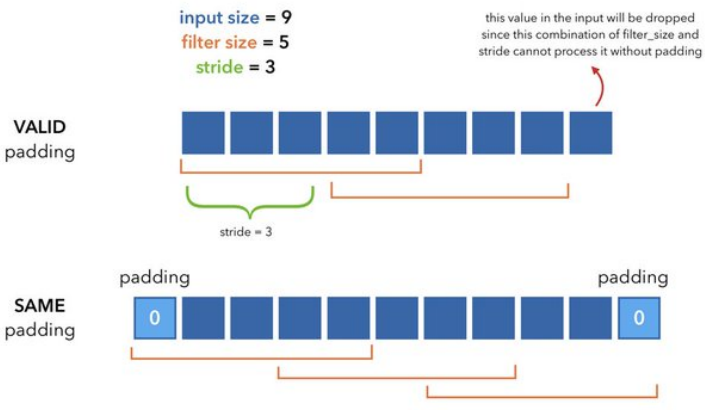
```python
from snorkel.labeling import labeling_function

@labeling_function()
def contains_tensorflow(text):
    condition = any(tag in text.lower() for tag in ("tensorflow", "tf"))
    return "tensorflow" if condition else None
```

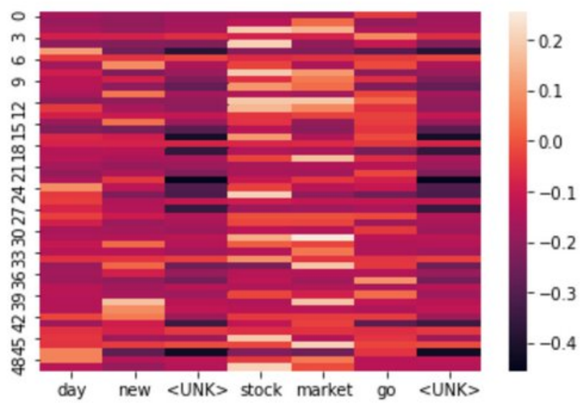
■ Modeling
- Baselines (simple → complex)
- Evaluation (overall, slices, generated)

- Experiment tracking (tracking, viewing and loading)
- Optimization (sampling + pruning)

■ These aren't just tutorial code snippets. We implement everything with clean and tested code.



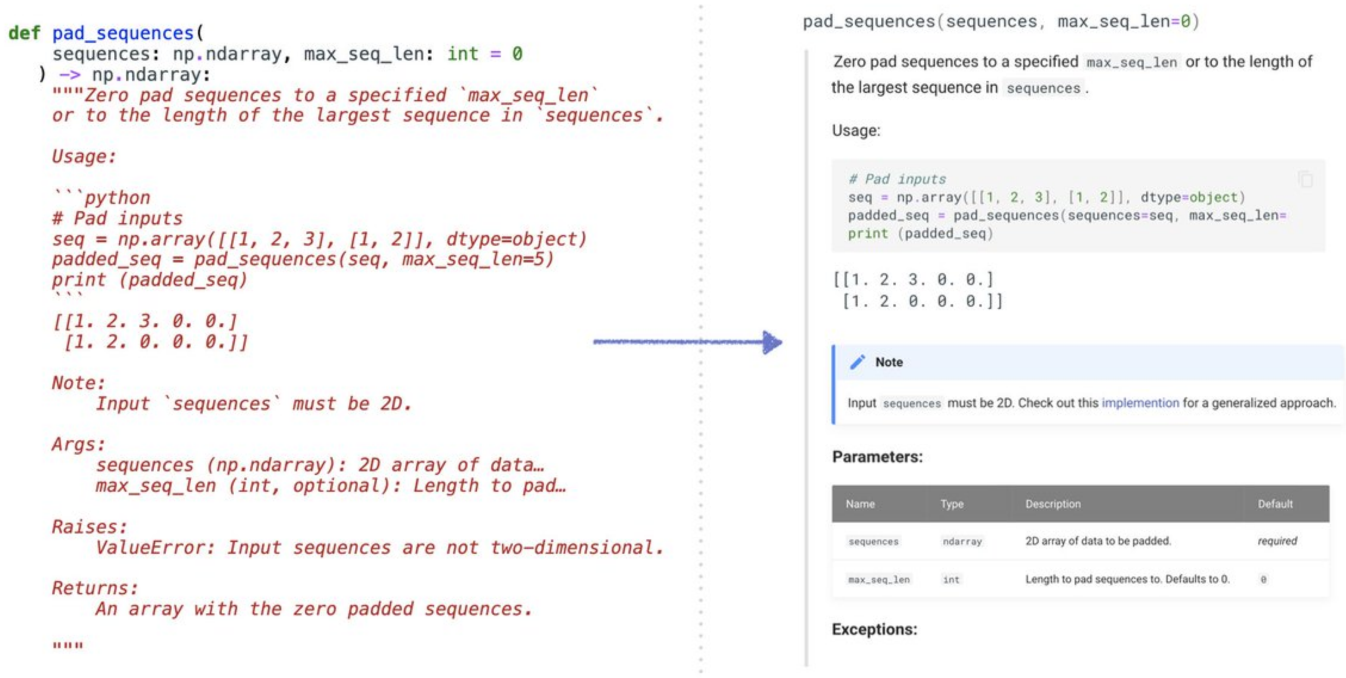




Next, we move our work from notebooks to scripts.

■ Scripting
- Organization
- Packaging (setup + virtualenv)
- Documentation (auto)
- Logging (logger, handler, formatter)
- Styling (black, isort, flake8)
- Makefile

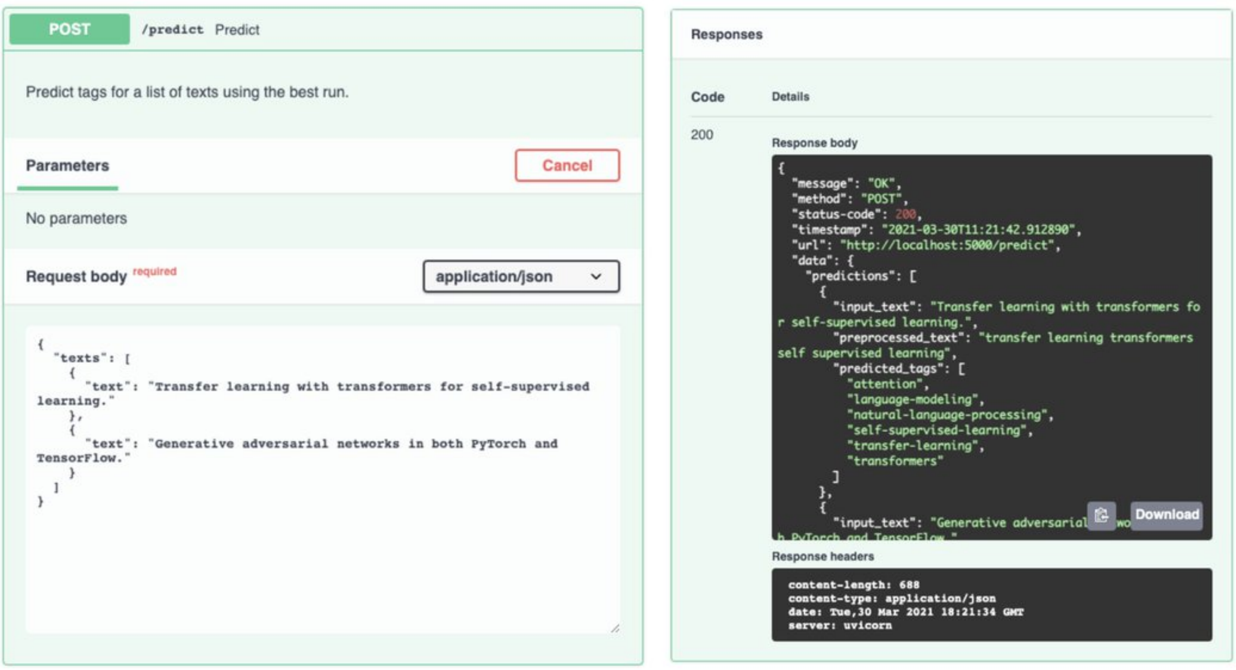
■ All of this makes for a very calm developing experience.



Now we're ready to wrap our application via various interfaces.

■ Interfaces
- Command-line (CLI)
- RESTful API with FastAPI (design, schemas, validation)

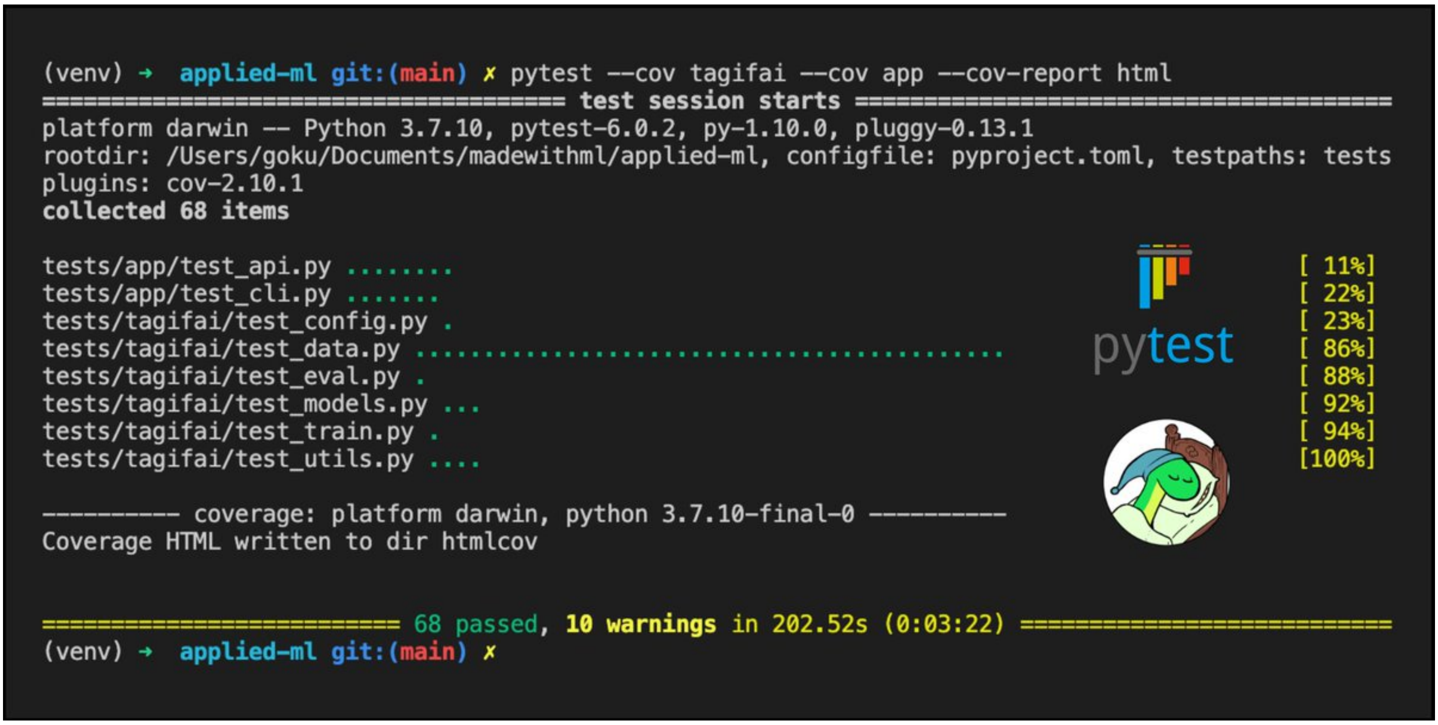
■ These interfaces allow us to quickly execute both internal (training, testing, etc.) and external (inference) tasks.



Throughout development, we've been testing not only our code but also our data and models.

■ Testing
- Test types, coverage, best practices

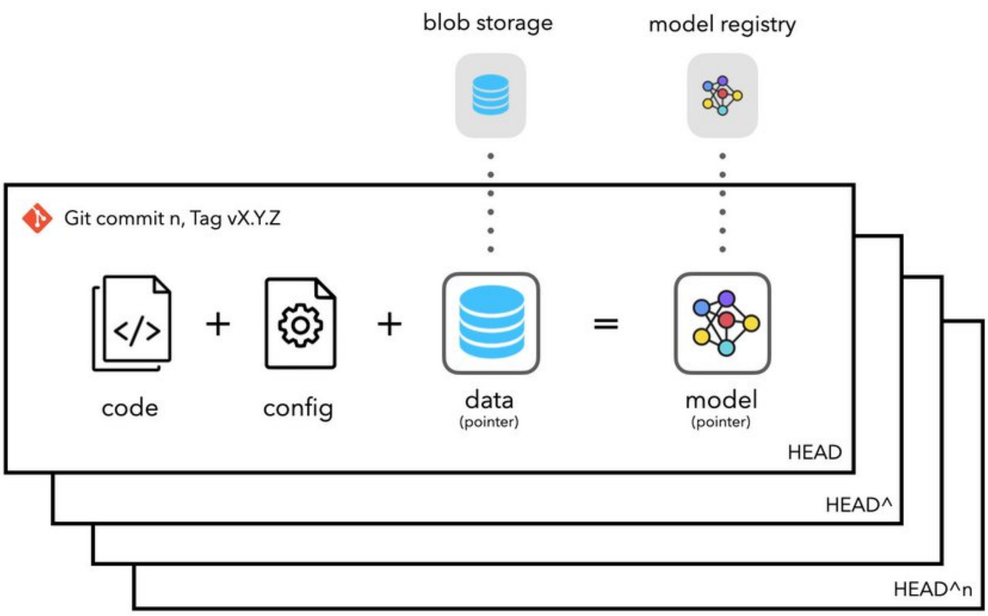
- Pytest fixtures, markers, parametrize
- Test data w/ Great Expectations
- Test models via slicing functions
- Behavioral tests



```
(venv) → applied-ml git:(main) ✗ pytest --cov tagifai --cov app --cov-report html
======================================== test session starts ========================================
platform darwin -- Python 3.7.10, pytest-6.0.2, py-1.10.0, pluggy-0.13.1
rootdir: /Users/goku/Documents/madewithml/applied-ml, configfile: pyproject.toml, testpaths: tests
plugins: cov-2.10.1
collected 68 items

tests/app/test_api.py ........                                                          [ 11%]
tests/app/test_cli.py .......                                                           [ 22%]
tests/tagifai/test_config.py .                                                          [ 23%]
tests/tagifai/test_data.py ........................................                     [ 86%]
tests/tagifai/test_eval.py .                                                            [ 88%]
tests/tagifai/test_models.py ...                                                        [ 92%]
tests/tagifai/test_train.py .                                                           [ 94%]
tests/tagifai/test_utils.py ....                                                        [100%]

---------- coverage: platform darwin, python 3.7.10-final-0 ----------
Coverage HTML written to dir htmlcov


=========================== 68 passed, 10 warnings in 202.52s (0:03:22) ===========================
(venv) → applied-ml git:(main) ✗
```

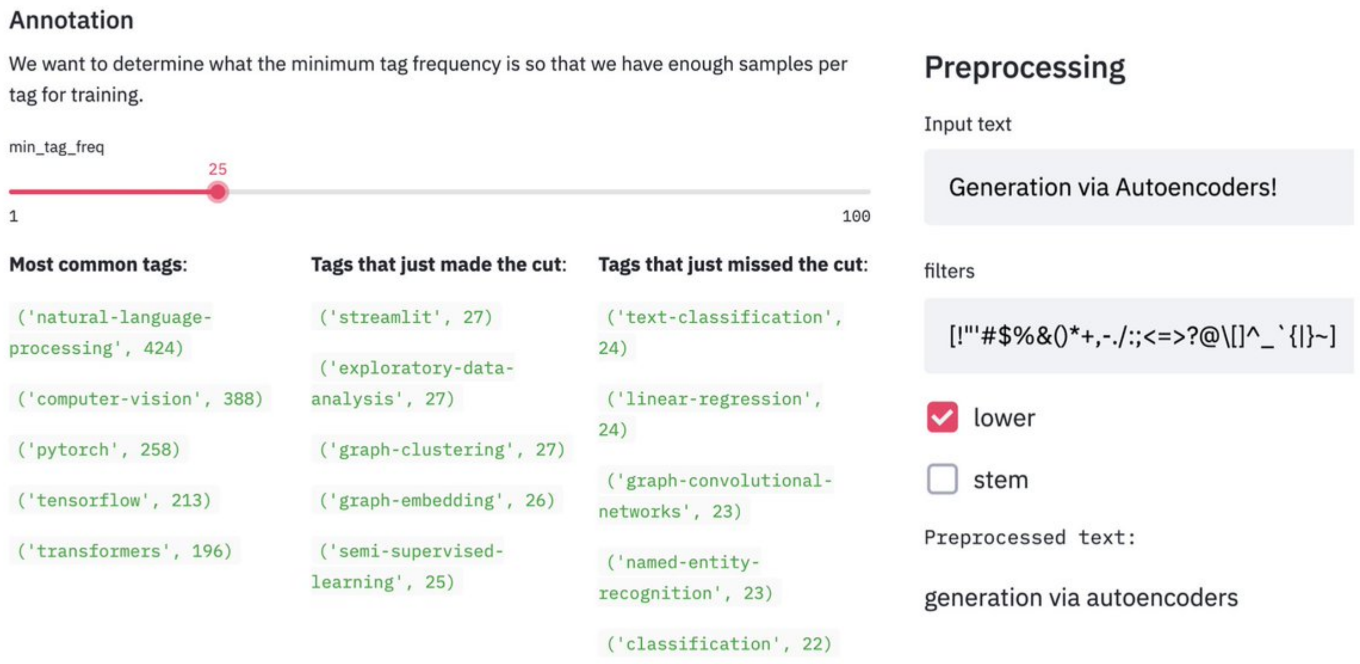

We want to ensure that our work is entirely reproducible by anyone.

■■ Reproducibility
- Git basics via workflows (dev, inspect, merge, etc.)
- Pre-commit hooks (+ custom local)
- Versioning code + config + data = models via DVC
- Containerization via Docker



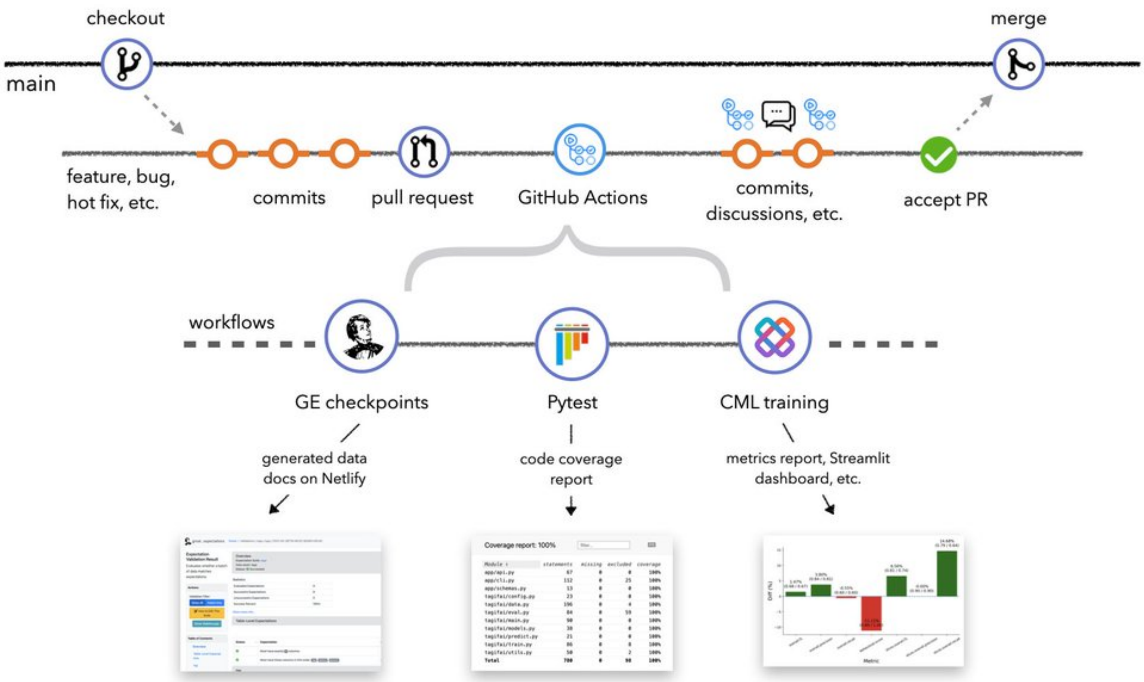
Next, we want to be able to showcase our work and enable interaction via @streamlit.

■ Dashboard:
- Data: annotation, EDA, preprocessing
- Performance: overall, slices, regressions
- Inference: intermediate & final outputs
- Inspection: labeling (FP), weaknesses (FN)



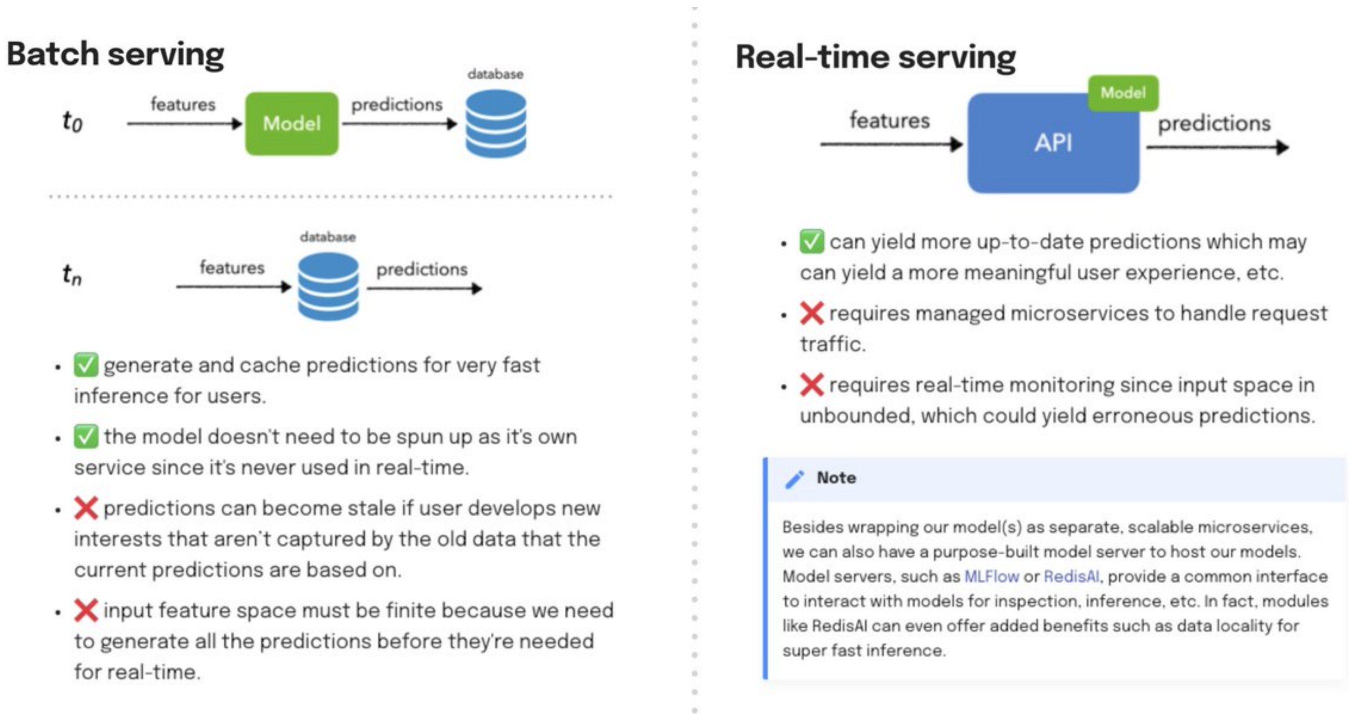
Then, we wrap all of the CI/CD workflows we've created with @GitHub Actions:

■ CI/CD workflows
- Workflow components (events, runners, jobs)
- Testing Actions locally using Act
- Best practices (ex. caching)
- ML Actions (Great Expectations checkpoints, DVC CML)



Next, we explore the infra needed to deploy & serve ML applications.

■■ Infrastructure:
- Serving (batch, real-time)
- Processing (batch, stream)
- Learning (offline, online)
- Testing (AB, canary, shadow)
- Optimization (prune, quantize, distill)
- Methods (K8s, serverless)



We ensure the health of our ML system with appropriate monitoring.

■ Monitoring:
- identifying drift (data, target, concept)
- measuring drift on uni/multivariate data via
- reducers (PCA, UAE)
- detectors (chi^2, KS, MMD)
- solutions (not always retraining)

Detecting drift as outlined in Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift

```python
from functools import partial
from alibi_detect.cd.pytorch import preprocess_drift
```

```python
# Untrained autoencoder (UAE) reducer
enc_dim = 32
reducer = nn.Sequential(
    embeddings_layer,
    nn.AdaptiveAvgPool2d((1, embedding_dim)),
    nn.Flatten(),
    nn.Linear(embedding_dim, 256),
    nn.ReLU(),
    nn.Linear(256, enc_dim)
).to(device).eval()
```

```python
# Preprocessing with the reducer
preprocess_fn = partial(preprocess_drift, model=reducer, batch_si
```

```python
from alibi_detect.cd import MMDDrift
```

```python
# Initialize drift detector
embeddings_mmd_drift_detector = MMDDrift(reference, backend="pytorch", p_va
```

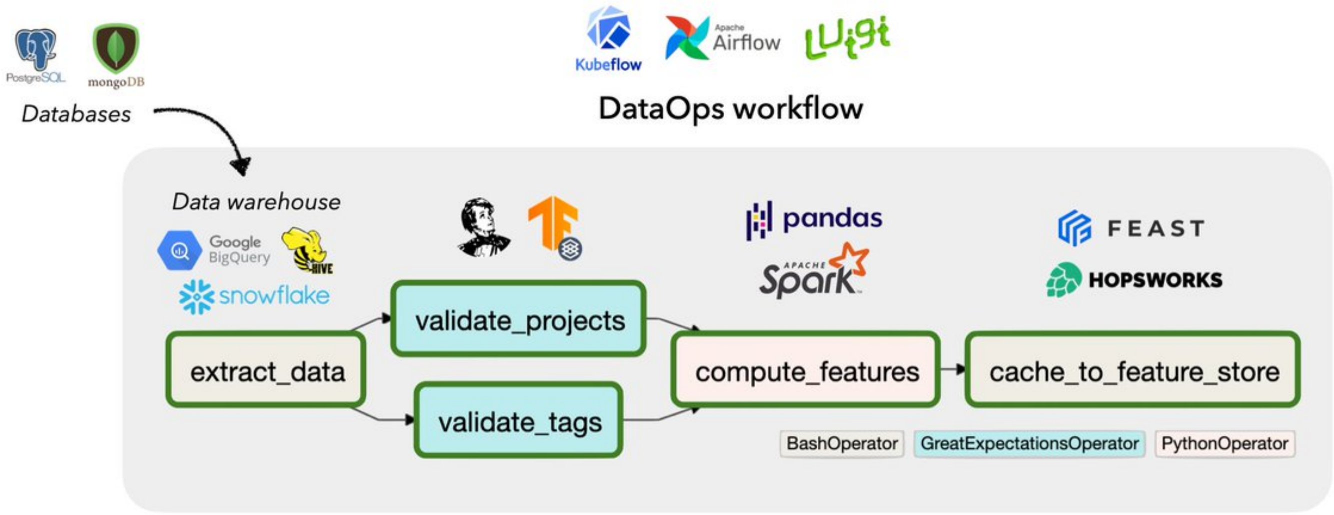
```python
# No drift
no_drift = get_data_tensor(texts=df.text[-200:].to_list())
embeddings_mmd_drift_detector.predict(no_drift)
```

```
{'data': {'is_drift': 0,
  'distance': 0.0006961822509765625,
  'p_val': 0.2800000011920929,
  'threshold': 0.01,
  'distance_threshold': 0.008359015},
 'meta': {'name': 'MMDDriftTorch',
  'detector_type': 'offline',
  'data_type': None,
  'backend': 'pytorch'}}
```

Finally, we connect our DataOps & MLOps workflows in our ML systems.

■■ Workflow orchestration w/ @ApacheAirflow
- DAGs
- Scheduler
- Tasks
- Operators
- Runs

■ Feature stores w/ @feast_dev
- data ingestion
- feature definitions
- historical/online features



Over the past 7 years, I've worked on ML and product at @Apple, health tech startups and ran my own venture in the rideshare space. I've worked with brilliant developers and managers and learned how to responsibly develop and iterate on ML systems across various industries.

I currently work closely with early-stage & mid-sized companies in helping them deliver value with ML while diving into the best & bespoke practices of this rapidly evolving space. I want to share that knowledge with the rest of the world so we can accelerate overall progress.

ML is not a separate industry, instead, it's a powerful way of thinking about data. The foundations we've laid out will continue to hold but the methods and avenues of application will evolve. So these lessons are by no means "complete" and we'll continue to keep them up-to-date.

Even more exciting content coming later this year, so stay tuned!

- ■ Among top MLOps repos on GitHub: https://t.co/gsYawqTq6U
- ■■ A highly recommended resource used by industry: https://t.co/pcMd8jZfo5
- ❤■ 30K+ community members: https://t.co/CswgmDZhCq