Advice on becoming a full-stack ML practitioner

Olivier Nguyen, Applied Research Scientist @ElementAI

October 28, 2019

About me - Olivier

- Applied Research Scientist @ Element AI
- Building AI products, integrate models in production
- Worked on applied research for document information extraction
- Previously
  - MAsc ECE @ University of Waterloo (2018)
  - Computer Engineering @ Concordia University (2016)

Agenda

Applied research in machine learning and deep learning

Trade-offs for code quality and rapid development, idea validation

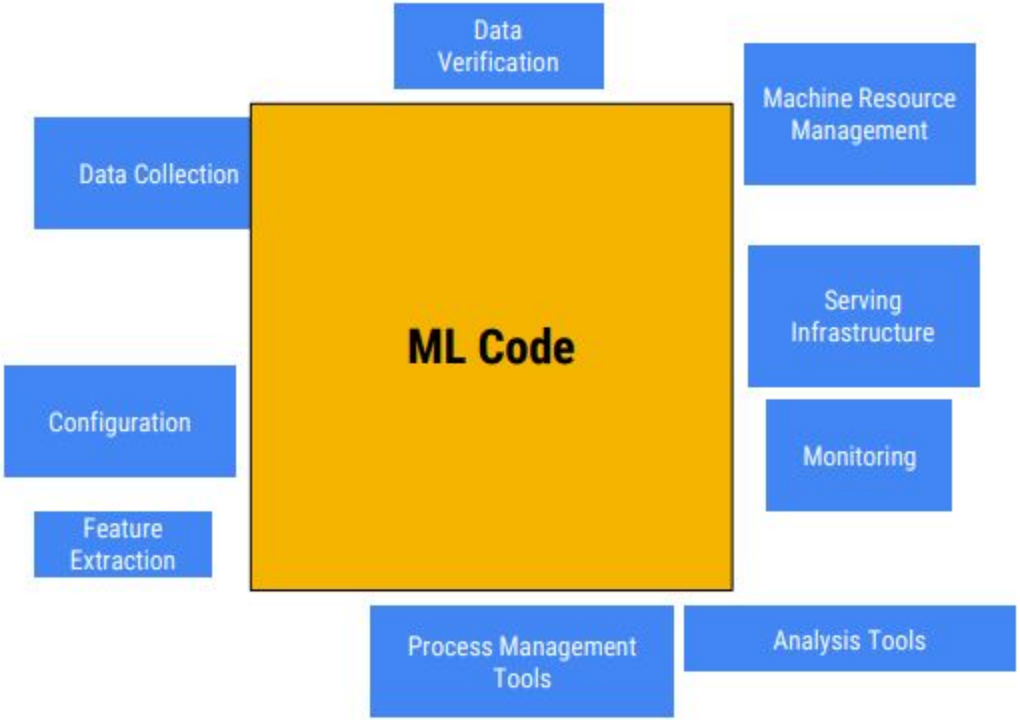
Research

- Write code quickly
- Run and iterate on experiments
- Compare various methods
- Demo webapps/proof of concepts

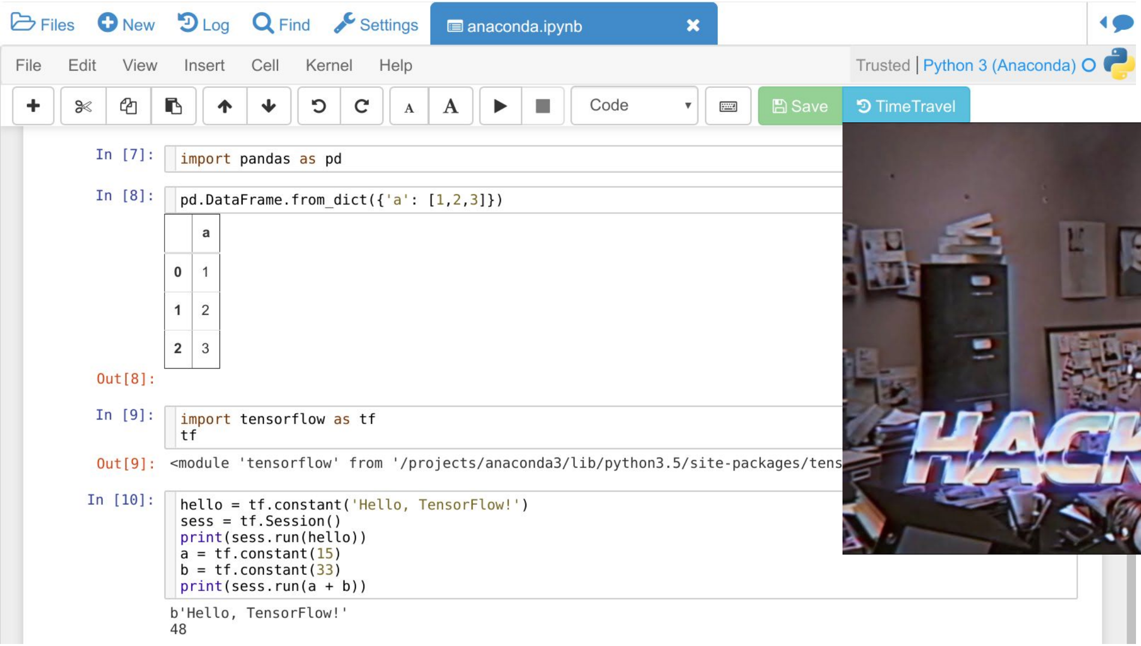
Products, libraries, reusable components

- Deploy models
- Track and measure performance
- Speed and efficiency
- Robust and reliable

Perception: ML Products are mostly about ML

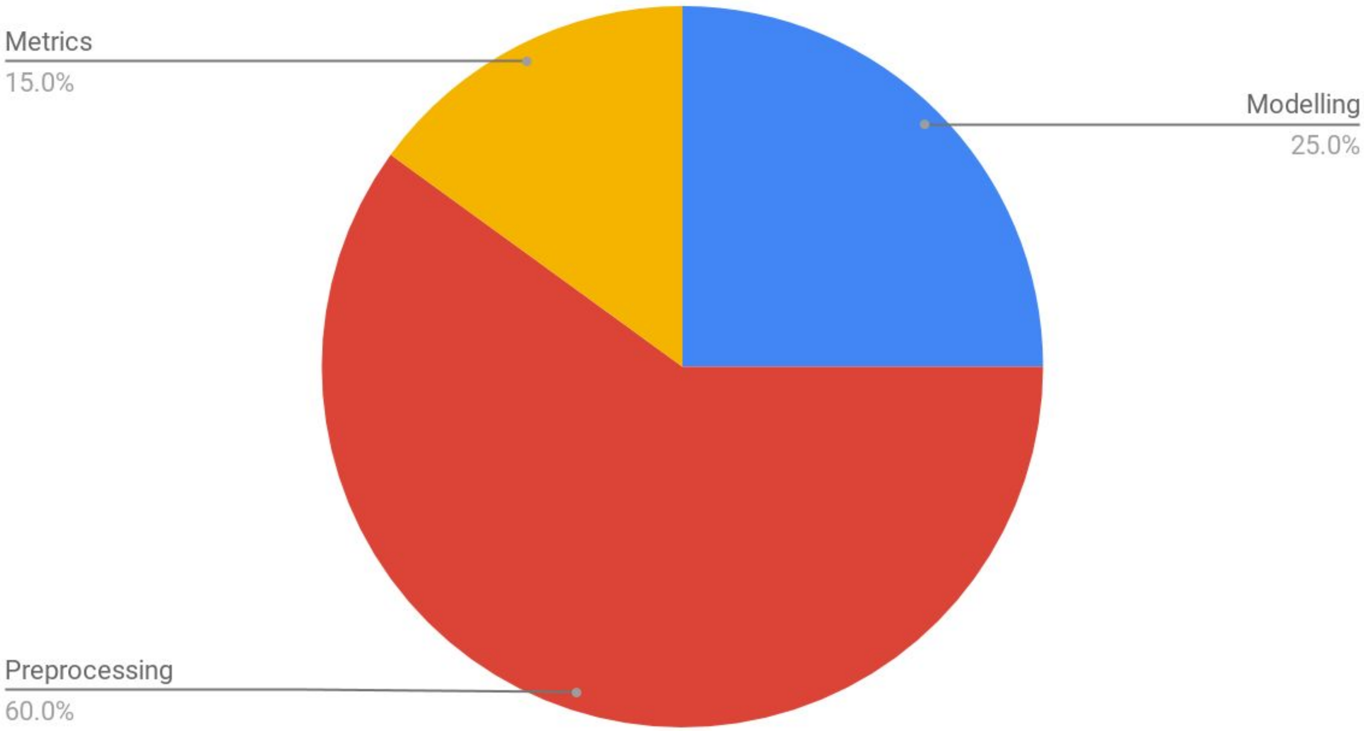


Source: [Building ML Products With Kubeflow (Kubecon 2018)]

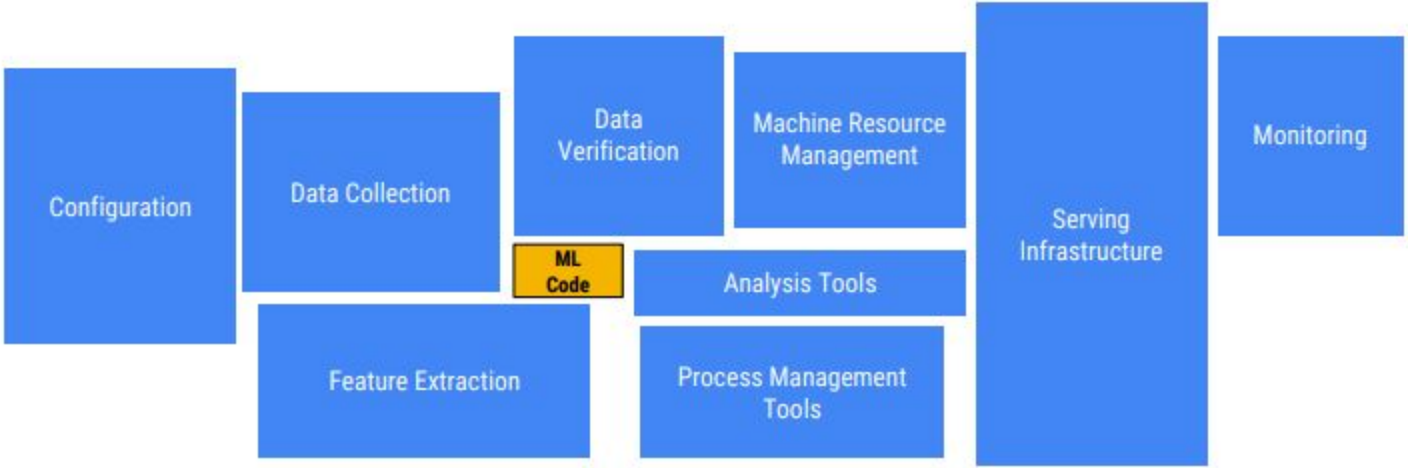

Files
New
Log
Find
Settings
anaconda.ipynb
File Edit View Insert Cell Kernel Help
Trusted
Python 3 (Anaconda)
Code
Save
TimeTravel
In [7]: import pandas as pd
In [8]: pd.DataFrame.from_dict({'a': [1,2,3]})
a
0 1
1 2
2 3
Out[8]:
In [9]: import tensorflow as tf
tf
Out[9]: <module 'tensorflow' from '/projects/anaconda3/lib/python3.5/site-packages/tens
In [10]: hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
a = tf.constant(15)
b = tf.constant(33)
print(sess.run(a + b))
b'Hello, TensorFlow!'
48


HACKERMAN

Also common expectation...



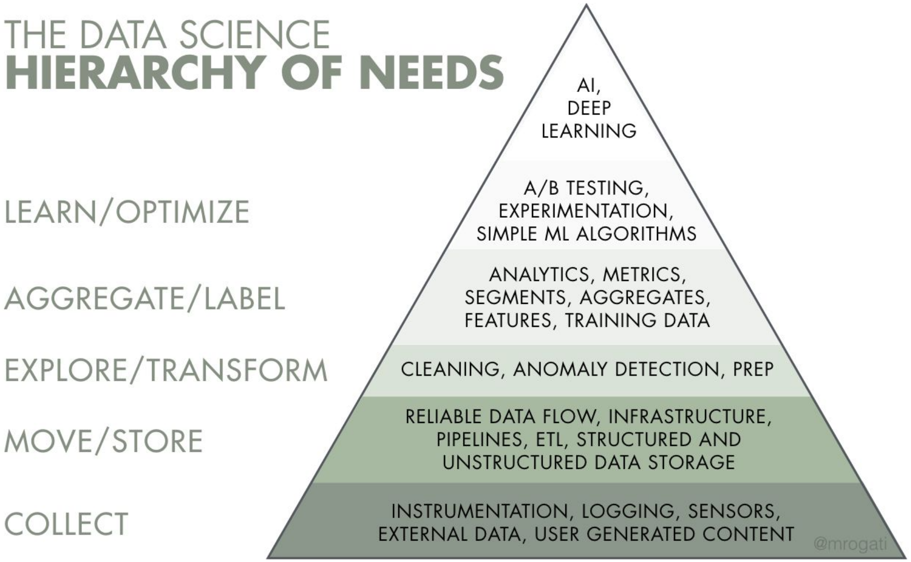
Reality: ML Requires DevOps; lots of it



Source: [Sculley et al.: Hidden Technical Debt in Machine Learning Systems]()

Machine learning ecosystem

- Huge amount of devops, data engineering
- Many companies are NOT AI ready



Source: The AI Hierarchy of Needs

Before you start throwing data to your models, check...

- Distribution of inputs
  - e.g. average sequence length, average pixel value etc.
- Distribution of outputs
  - e.g. class imbalances
- Dataset-specific measures
- If you're in for the long run, write specific visualization tools
- Is your data good enough?
  - https://petewarden.com/2018/05/28/why-you-need-to-improve-your-training-data-and-how-to-do-it/

Start as simple as possible - Linear models

- Know when a deep neural network is needed
- Linear models are:
  - Fast to train
  - Easy to deploy
  - Easy to interpret
- Gets a baseline and your entire pipeline setup
  - Compare complicated models to baseline
- Feature engineering helps you know your data
  - Writing rules/heuristics sucks, but it can help you figure out what you want the model to learn
- More likely to stay in production compared to a DL model

Use sklearn Pipelines

```
feature_extractor = sklearn.pipeline.FeatureUnion([
      ("tfidf_token_ngrams", TfidfVectorizer(ngram_range=(1,2), lowercase=False, stop_words='english')),
      ('char_len', CharLengthExtractor()),
      ('num_words', NumWordExtractor())
])

lr_tfidf = sklearn.pipeline.Pipeline([
    ('feature_extraction', feature_extractor),
    ('logistic_regression', GridSearchCV(
            LogisticRegression(penalty='l2'), param_grid=params))
])

X: List[str] = newsgroups_train.data
y: List[int] = newsgroups_train.target

scores = cross_val_score(lr_tfidf, X, y, cv=5, n_jobs=-1)
```

Starting simple - Deep learning models

- Don’t worry about code duplication - Refactor later!
- Overfit on your training set
    - Can the network memorize the dataset?

```
train_dataset = datasets.ImageFolder(root='image_data/train', transform=data_transform)

train_loader = torch.utils.data.DataLoader(train_dataset,
                                           batch_size=4,
                                           pin_memory=True,
                                           num_workers=4)

valid_loader = torch.utils.data.DataLoader(train_dataset[:100],
                                           batch_size=4,
                                           pin_memory=True,
                                           num_workers=4)
```

Use model configs

- Use configs to define your experiments
- Save them on disk during training

model_config.json

```
$ python train.py -c model_config.json
```

```json
{
    "epochs": 100,
    "checkpoint_path": "/ckpts/"
    "model": {
        "embedding": {
          "glove": {
            "embedding_dim": 100
          },
          "seq_representation": {
            "LSTM": {
              "dropout": 0.5
            }
          }
        },
        "decoder": {
          "hidden_dims": [
            100
          ]
        }
    }
}
```

```json
{
    "epochs": 100,
    "checkpoint_path": "/ckpts/"
    "model": {
        "embedding": {
          "glove": {
            "embedding_dim": 100
          },
          "seq_representation": {
            "GRU": {
              "dropout": 0.5
            }
          }
        },
        "decoder": {
          "hidden_dims": [
            100
          ]
        }
    }
}
```

Running experiments

- State a hypothesis clearly and design experiment before launching jobs
- Use version control for your experiments
- Run controlled experiments (Test only one thing at a time)

Experimenter	git SHA	Background Search Method	Model	Dataset	Train Acc	Validation Acc	Notes
Pradeep	fc8d6ca3	Lucene	QAMNS (50d)	Intermediate	0.3114	0.3045	patience=20
Pradeep	fc8d6ca3	Lucene	QAMNS (300d)	Intermediate	0.8317	0.3864	patience=20
Pradeep	fc8d6ca3	BOW-LSH question+answers Glove 50d	QAMNS (50d)	Intermediate	0.3008	0.35	patience=20
Pradeep	fc8d6ca3	BOW-LSH question+answers Glove 50d	QAMNS (300d)	Intermediate	0.7466	0.4227	patience=20

Log as much as you can

- You should be logging your inputs, not just your outputs
- You should know
  - How many examples there are
  - How many batches that corresponds to
  - How many batches constitute an epoch

```
2019-10-09T16:03:50.753265Z [info     ] ------ Epoch 63 ------
2019-10-09T16:04:27.073436Z [info     ] > [Training] Detection Loss: 0.018
2019-10-09T16:04:28.656882Z [info     ] Post Metrics ok                trial_id=963377
100%|██████████| 96/96 [00:08<00:00, 11.35it/s, Detection Loss=0.0869]
Validation batch time
 > total: 2407.14 ms
 > input        : 1242.78 ms
 > to_device    : 118.93 ms
 > model        : 76.29 ms
 > loss         : 968.16 ms
 > other        : 0.98 ms
2019-10-09T16:04:37.502948Z [info     ] > [Validation] Detection Loss: 0.087
2019-10-09T16:04:37.508694Z [info     ] Early stopping...
2019-10-09T16:04:37.519304Z [info     ] Loading checkpoint from <_io.BufferedReader name='/checkpoints/loss=0.001.pth'>
2019-10-09T16:04:37.650925Z [info     ] Post Metrics ok                trial_id=963377
2019-10-09T16:04:37.691983Z [info     ] Checkpoint loaded, serializing...
2019-10-09T16:04:43.875089Z [info     ] Serialized to /checkpoints/serialized_model.pklz
2019-10-09T16:04:43.877709Z [info     ] The experiment succeeded!
```

Keep an eye on the GPU utilization

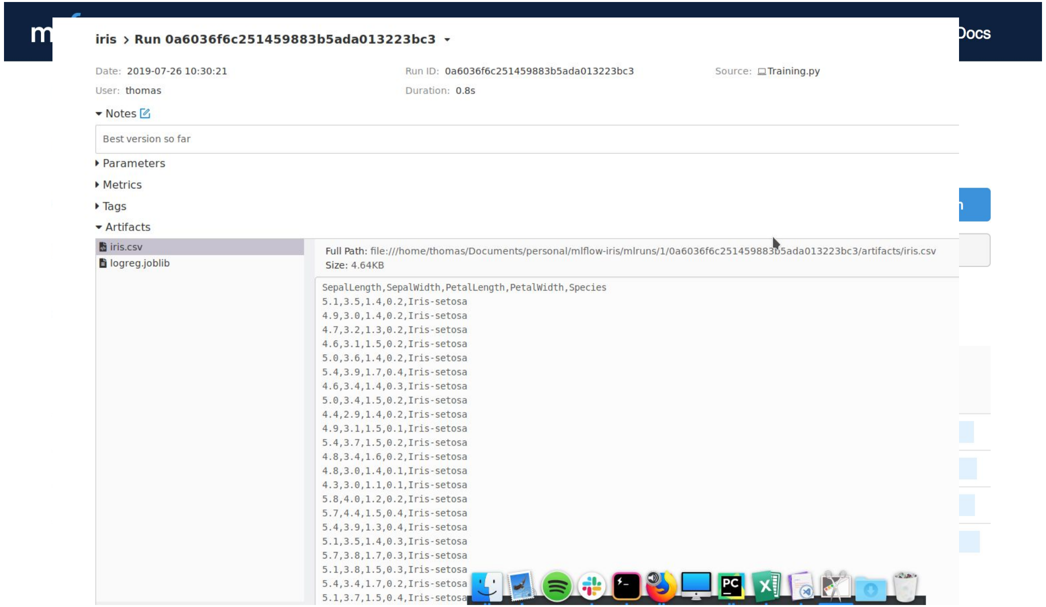
Structure your checkpoint directories

- Group your artifacts under the same place

```
checkpoints/
├── weights/
│   ├── loss=0.03.pth
│   └── loss=0.005.pth
├── config.yaml
├── predictions.json
├── serialized_model.pklz
```

Experiment tracking tools

- Need to reproduce, track and measure model development
- Check out MLFlow
- Also: Sacred, Neptune.ml, wandb, comet.ml



Training pipeline - Keep it simple!

- Avoid rewriting code; leverage open-source libraries
- For PyTorch, check out Lightning, Fast.ai, Ignite

```
for epoch in range(num_epochs):
    for batch in trainloader:
        inputs, labels = batch
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

```
model = CoolModel()
exp = Experiment(save_dir=os.getcwd())

trainer = Trainer(experiment=exp,
                  max_nb_epochs=1,
                  train_percent_check=0.1)

trainer.fit(model)
```

Document and test your DL models

- Comment for tensor shapes
- Use unit tests while implementing your model

```
def dot_product_attention(q, k, v, bias, dropout, return_weights=False):
    d_k = q.size(-1)
    # [batch, length, hidden_dim] x [batch, hidden_dim, length] -> [batch, length, length]
    logits = torch.bmm(q, k.transpose(1, 2).contiguous())

    # Scale the keys to prevent large values and vanishing gradients
    # See 3.1.1 in Attention is All You Need
    logits /= math.sqrt(d_k)

    if bias is not None:
        logits += bias
    size = logits.size()
    weights = F.softmax(logits.view(size[0] * size[1], size[2]), dim=1)
    weights = F.dropout(weights, dropout)

    # [batch_size, length, length]
    weights = weights.view(size)

    # [batch_size, length, hidden_dim]
    return torch.bmm(weights, v), weights
```

Testing deep learning models

- Use test fixtures - Keep a subset of the data in a repo and run tests on them.

```
.
└── tests/
    └── fixtures/
        └── dataset/
            ├── labels.json
            ├── train/
            │   ├── img-1.png
            │   └── img-2.png
            └── valid/
                ├── img-3.png
                └── img-4.png
```

```
_TEST_DATASET_DIR = Pathlib("tests/fixtures/dataset")

@pytest.fixture
def dataset():
    labels = json.load(open(_TEST_DATASET_DIR / "labels.json"))
    train_files = glob.glob(_TEST_DATASET_DIR / "train/*")
    valid_files = glob.glob(_TEST_DATASET_DIR / "valid/*")
    return dict(labels=labels,
                train_files=train_files,
                valid_files=valid_files)

def test_preprocess_dataset(dataset):
    ...
```

Error analysis loop

- Look at your test set examples
  - What is the best example?
  - What is the worse example?
- Jupyter notebook with annotations
- Dump predictions after benchmarking or after training

```
df = json.load(open("predictions.json"))
error_analysis_entries = []
for idx, row in df.sample(5).iterrows():
    entry = row.copy()
    plt.imshow(entry[“filename”])
    entry["notes"] = input("Enter notes about this example")
    error_analysis_entries.append(entry)

all_entries_df = pd.DataFrame(error_analysis_entries)
```

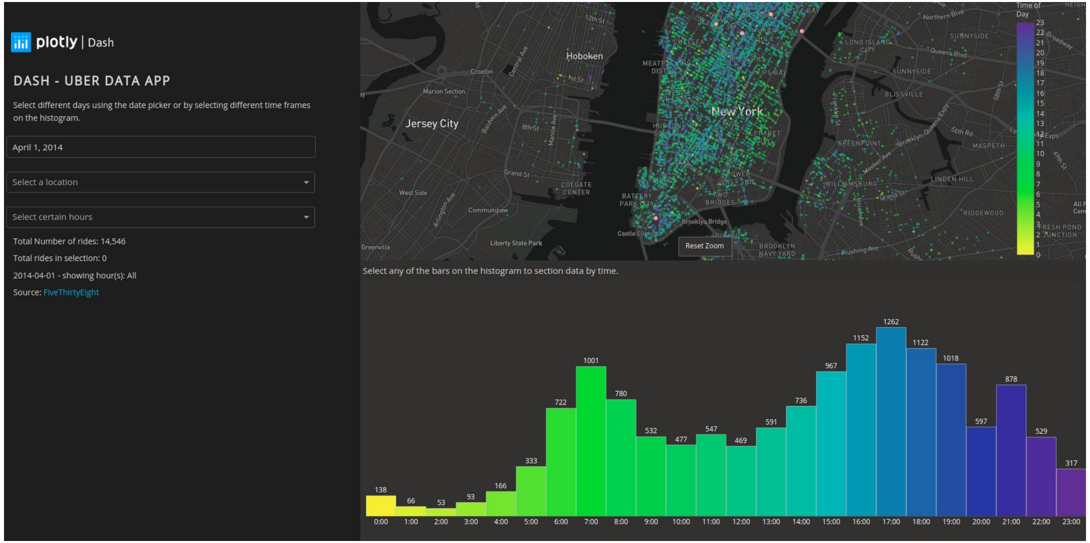
	filenames	predictions	label	probs	notes
16	img16.png	0	1	0.16	Cat appears to be obstructed
1	img1.png	1	1	0.01	Low quality image
84	img84.png	0	1	0.84	Dog looks very much like a cat
11	img11.png	1	1	0.11	There is no cat
45	img45.png	1	1	0.45	This cat looks different

Some gotchas

- Lowest loss does not necessarily mean best performance
  - Your model might learn things you don't want it to, and it will benchmark well
  - Monitor performance with other metrics than the loss
- Make sure you have the same preprocessing steps at test time
- Mixing up your dimensions
  - [batch, width, height, channels] vs. [batch, channels, width, height]
  - [batch, seq_len, embedding_size] vs. [seq_len, batch, embedding_size]

Deploy your model on a webapp - Dash

- Easy-to-use libraries like [plotly.Dash]() make it super easy to deploy dashboard webapps in Python
- Importing your models is easy
- Visualize and test examples live
- Learn to deploy models



Model explainability - Tips and tricks

- Plot errors vs. individual features
  - E.g. am I doing well for certain parts of the input space and poorly for others?
- Consider ablating features to check whether you get expected results
  - LIME
  - Remove components and measure performance

Model	Dev. Accuracy
Full model	42.7
token features, no similarity	28.1
all features, no similarity	37.8
similarity only, no features	27.5

Table 3: Development accuracy of ablated parser variants trained without parts of the entity linking

Use docker containers

- Run code anywhere
- Make it easier to reproduce and share code

Questions?

Slides: olinguyen.com
olivier.nguyen@elementai.com