Machine Learning for CI

Kyra Wulffert
kwulffert@yahoo.com

Matthew Treinish
mtreinish@kortar.org

Andrea Frittoli
andrea.frittoli@gmail.com

https://github.com/afrittoli/ciml_talk
CI at Scale

- Continuous Integration
- Continuous Log Data
- Lots of data, little time
- Triaging failures?
- AI to the rescue!

Source: subunit2sql-graph dailycount
The OpenStack use case

- Integration testing in a VM
- System logs, application logs
- Dstat data
- Gate testing
- Not only OpenStack

Normalized system average load for different examples
Collecting data

- Automation and repeatability
- Light-weight data validation
- Object storage for data
- Periodic Action on OpenWhisk
Experiment Workflow

- Visualize data
- Define a dataset
- Define an experiment
- Run the training
- Collect results
- Visualize data

# Build an s3 backed dataset
```bash
ciml-build-dataset --dataset cpu-load-1min-dataset \
  --build-name tempest-full \
  --slicer :2000 \
  --sample-interval 10min \
  --features-regex "(usr|1min)" \
  --class-label status \
  --tdt-split 7 0 3 \
  --data-path s3://cimlrawdata \
  --target-data-path s3://cimldatasets
```

Dataset preparation diagram
Data Selection

- What is dstat data?
- Experiment reproducibility
- Dataset selection
  - Dstat feature selection
  - Data resolution (down-sampling)

Sample of dstat data

<table>
<thead>
<tr>
<th>time</th>
<th>usr</th>
<th>used</th>
<th>writ</th>
<th>lm</th>
</tr>
</thead>
<tbody>
<tr>
<td>16/03/2018 21:44:52</td>
<td>6.1</td>
<td>7.36 (\times) 10^8</td>
<td>5.78 (\times) 10^5</td>
<td>0.97</td>
</tr>
<tr>
<td>16/03/2018 21:44:53</td>
<td>7.45</td>
<td>7.43 (\times) 10^8</td>
<td>3.6 (\times) 10^5</td>
<td>0.97</td>
</tr>
<tr>
<td>16/03/2018 21:44:54</td>
<td>4.27</td>
<td>7.31 (\times) 10^8</td>
<td>4.01 (\times) 10^5</td>
<td>0.97</td>
</tr>
<tr>
<td>16/03/2018 21:44:55</td>
<td>1</td>
<td>7.43 (\times) 10^8</td>
<td>4.096</td>
<td>0.97</td>
</tr>
<tr>
<td>16/03/2018 21:44:56</td>
<td>0.5</td>
<td>7.44 (\times) 10^8</td>
<td>1.5 (\times) 10^7</td>
<td>0.97</td>
</tr>
<tr>
<td>16/03/2018 21:44:57</td>
<td>1.75</td>
<td>7.31 (\times) 10^8</td>
<td>4.096</td>
<td>0.97</td>
</tr>
<tr>
<td>16/03/2018 21:44:58</td>
<td>0.88</td>
<td>7.43 (\times) 10^8</td>
<td>4.096</td>
<td>0.9</td>
</tr>
<tr>
<td>16/03/2018 21:44:59</td>
<td>1.39</td>
<td>7.31 (\times) 10^8</td>
<td>4.51 (\times) 10^5</td>
<td>0.9</td>
</tr>
<tr>
<td>16/03/2018 21:45:00</td>
<td>1.01</td>
<td>7.44 (\times) 10^8</td>
<td>4.096</td>
<td>0.9</td>
</tr>
<tr>
<td>16/03/2018 21:45:01</td>
<td>0.75</td>
<td>7.46 (\times) 10^8</td>
<td>61.440</td>
<td>0.9</td>
</tr>
<tr>
<td>16/03/2018 21:45:02</td>
<td>1.26</td>
<td>7.31 (\times) 10^8</td>
<td>4.096</td>
<td>0.9</td>
</tr>
<tr>
<td>16/03/2018 21:45:03</td>
<td>1.13</td>
<td>7.44 (\times) 10^8</td>
<td>4.096</td>
<td>0.82</td>
</tr>
<tr>
<td>16/03/2018 21:45:04</td>
<td>5.77</td>
<td>7.77 (\times) 10^8</td>
<td>1.72 (\times) 10^5</td>
<td>0.82</td>
</tr>
<tr>
<td>16/03/2018 21:45:05</td>
<td>9.85</td>
<td>8.31 (\times) 10^8</td>
<td>4.99 (\times) 10^6</td>
<td>0.82</td>
</tr>
<tr>
<td>16/03/2018 21:45:06</td>
<td>3.88</td>
<td>8.46 (\times) 10^8</td>
<td>8.25 (\times) 10^7</td>
<td>0.82</td>
</tr>
</tbody>
</table>
Data Normalization

- **Unrolling**

  Sample of unrolled data

<table>
<thead>
<tr>
<th>usr1</th>
<th>usr2</th>
<th>usr3</th>
<th>1m1</th>
<th>1m2</th>
<th>1m3</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>1.75</td>
<td>1.26</td>
<td>0.97</td>
<td>0.97</td>
<td>0.9</td>
</tr>
<tr>
<td>5.9</td>
<td>1.5</td>
<td>3.1</td>
<td>0.9</td>
<td>0.92</td>
<td>0.97</td>
</tr>
<tr>
<td>5.8</td>
<td>1.76</td>
<td>2.2</td>
<td>0.89</td>
<td>0.91</td>
<td>0.94</td>
</tr>
</tbody>
</table>

- **Normalizing**

  Sample of normalized data

<table>
<thead>
<tr>
<th>usr1</th>
<th>usr2</th>
<th>usr3</th>
<th>1m1</th>
<th>1m2</th>
<th>1m3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>0.3</td>
<td>-0.5</td>
<td>0.6</td>
<td>0.6</td>
<td>-0.5</td>
</tr>
<tr>
<td>-0.1</td>
<td>-0.7</td>
<td>0.5</td>
<td>-0.3</td>
<td>-0.2</td>
<td>0.5</td>
</tr>
<tr>
<td>-0.4</td>
<td>0.3</td>
<td>0</td>
<td>-0.4</td>
<td>-0.4</td>
<td>0</td>
</tr>
</tbody>
</table>
Building the dataset

- Split in training, dev, test
- Obtain classes
- Store normalized data on s3
- Input function for training
- Input function for evaluation

Structure of a dataset
Experiment Workflow

- Visualize data
- Define a dataset
- Define an experiment
- Run the training
- Collect results
- Visualize data

# Define a local experiment
```
ciml-setup-experiment --experiment dnn-5x100 
  --estimator tf.estimator.DNNClassifier 
  --hidden-layers 100/100/100/100/100 
  --steps $(( 2000 / 128 * 500 )) 
  --batch-size 128 
  --epochs 500 
  --data-path s3://cimldatasets
```

# Train the model locally based on the dataset and experiment
```
ciml-train-model --dataset cpu-load-1min-dataset 
  --experiment dnn-5x100 
  --data-path s3://cimldatasets
```

# Train the same model in a FfDL cluster
```
ffdl_train.sh cpu-load-1min-dataset dnn-5x100
```
Training Infrastructure

- TensorFlow Estimator API
- CIML wrapper
- ML framework interchangable
- Training Options:
  - Run on a local machine
  - Helm deploy CIML, run in containers
  - Submit training jobs to Ffdl
  - Kubeflow

Prediction

- Event driven: near real time
- No request to serve the prediction to
- MQTT Trigger from the CI system
- CIML produces the prediction
- Trusted Source: Continuous Training

- CIML kubernetes app components:
  - MQTT Client receives events
  - Data module fetches and prepares data
  - TensorFlow wrapper issues the prediction
  - Example: comment back on Gerrit/Github
DNN - Binary Classification

- Classes: Passed or Failed
- Supervised training
- TensorFlow `DNNClassifier`, classes=2
- Dataset:
  - CI Job "tempest-full"
  - Gate pipeline only
  - 3000 examples, 2100 training, 900 test
- Hyper-parameters:
  - Activation function: ReLU
  - Output layer: Sigmoid
  - Optimizer: Adagrad
  - Learning rate (initial): 0.05
  - 5 hidden layers, 100 units per layer
  - Batch Size: 128, Epochs: 500
DNN - Binary Classification

- Selecting the best feature set
- Primary metric: accuracy
- Aim for lower loss, caveat: overfitting
- Key:
  - \texttt{usr}: User CPU
  - \texttt{used}: Used Memory
  - \texttt{1m}: System Load - 1min Average
  - Data Resolution: 1\texttt{min}
  - Source: TensorFlow evaluation

- Winner: (\texttt{usr}, 1\texttt{m}) tuple
- Accuracy achieved: \textbf{0.992}
- 7 mistakes on a 900 test set
DNN - Binary Classification

- Selecting the data resolution
- Primary metric: accuracy
- Aim for lower loss, caveat: overfitting
- Note: careful with NaN after down-sampling
- Key:
  - Original data frequency: 1s
  - x-axis: new sampling rate
  - Features: (**usr**, **1m**)
  - Source: TensorFlow evaluation
- Winner: 10s
- Accuracy achieved: 0.993
- 7 mistakes on a 900 test set
Changing test job

- Train with "tempest-full"
- Evaluating with "tempest-full-py3"
  - Similar setup, uses python3
  - It does not include swift and swift tests
  - 600 examples evaluation set

- Dataset and training setup:
  - Features: (usr, 1m)
  - Resolution: 1min
  - Same hyper-parameters
Binary Classification - Summary

- User CPU and 1min Load Avg
- Resolution: 10s best, 1 minute may be enough
- High accuracy: 0.993
- High auc_precision_recall: 0.945
- A trained model might be applicable to similar CI jobs
DNN - Multi Class

- Classes: Hosting Cloud Provider
- Supervised training
- TensorFlow **`DNNClassifier`, classes=10**
- Dataset:
  - CI Job "tempest-full"
  - Gate pipeline only
  - 3000 examples, 2100 training, 900 test
- Hyper-parameters:
  - Activation function: ReLU
  - Output layer: Sigmoid
  - Optimizer: Adagrad
  - Learning rate (initial): 0.05
  - 5 hidden layers, 100 units per layer
  - Batch Size: 128, Epochs: 500
DNN - Multi Class

- Features: (usr, 1m)
- Resolution: 1min
- Loss converges, but...
- Evaluation accuracy achieved: \textbf{0.601}
- Not good!
Multi Class - Different Features

- Try different combinations of features
- Primary metric: accuracy
- Aim for lower loss, caveat: overfitting
- Key:
  - `usr`: User CPU
  - `used`: Used Memory
  - `1m`: System Load - 1min Average
- Data Resolution: 1min
- Source: TensorFlow evaluation output
- No real improvement
- Best accuracy achieved: 0.603
- Adding Disk I/O or process data does not help either
Multi Class - Changing Resolution

▶ Trying to change the data resolution
▶ Primary metric: accuracy
▶ Aim for lower loss, caveat: overfitting
▶ Key:
  ▶ Original data frequency: 1s
  ▶ x-axis: new sampling rate
  ▶ Features: (usr, 1m)
  ▶ Source: TensorFlow evaluation
▶ No real improvement
▶ Best accuracy achieved: 0.624
Multi Class - Network topology

- Trying to change the network depth
- Trying to change number of units per layer
- Primary metric: accuracy
- Aim for lower loss, caveat: overfitting
- Key:
  - x-axis: units and hidden layers
  - Features: \((\text{usr}, 1\text{m})\)
  - Resolution: \(1\text{min}\)
  - Source: TensorFlow evaluation
- No real improvement
- Best accuracy achieved: \(0.668\)
Multi Class - Reducing the number of classes

- Reducing the number of classes
  - Different regions from a Cloud Operator
  - Consider as a single class
  - New number of classes is 6

- Experiments:
  - Train with different feature sets
  - Train with different resolutions
  - Source: TensorFlow evaluation

- Significant improvement!
- Best accuracy achieved: 0.902
- What does that mean?
Multi Class - Tuning network topology

- Tuning network topology
- Experiments:
  - x-axis: units and hidden layers
  - Features: (usr, 1m)
  - Resolution: 1min
- Some improvement
- Winner: 3x100. Accuracy: 0.925
Multi Class - Changing test job

- Train with "tempest-full"
- Evaluating with "tempest-full-py3"
  - Similar setup, uses python3
  - It does not include swift and swift tests
  - 600 examples evaluation set
- Dataset and training setup:
  - Features: (usr, 1m)
  - Resolution: 1min
  - Same hyper-parameters (dnn-3x100)

<table>
<thead>
<tr>
<th>metric</th>
<th>tempest-full</th>
<th>tempest-full-py3</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy</td>
<td>0.925</td>
<td>0.775</td>
</tr>
<tr>
<td>average_loss</td>
<td>0.978</td>
<td>3.271</td>
</tr>
<tr>
<td>loss</td>
<td>586.713</td>
<td>1,962.447</td>
</tr>
</tbody>
</table>
Multi Class - Summary

- User CPU and 1min Load Avg
- Resolution: 1 minute is enough
- Hyperparameters: 3 hidden layers, 100 units each
- Reasonable accuracy: \(0.925\)
- A trained model is not applicable to similar CI jobs

Training Loss - usr/1m, 1min, dnn3x100 - Source: TensorBoard
Conclusions

- Collect data
- Know your data
- Work with cloud tools
- Able to confirm that system load plays a role in failures
- Load profiles are consistent across regions in our cloud providers
Future Work

- Build a service with persistence to track experiments over time
- Look at adapting techniques for new models with different data
- Human curated dataset for supervised training
- Research clustering techniques for unsupervised training
- Explore job portability
Questions?

- This talk: https://github.com/afrittoli/ciml_talk
- CIML: https://github.com/mtreinish/ciml