Machine Learning for CI

Kyra Wulffert
kwulffert@yahoo.com

Matthew Treinish
mtreinish@kortar.org

Andrea Frittoli
andrea.frittoli@gmail.com

August 29, 2018

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

Data caching diagram
Experiment Workflow

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

```bash
# Build an s3 backed dataset

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
Experiment Workflow

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

```bash
# Define a local experiment
```
ciml-setup-experiment --experiment dnn--5x100 \
--dataset cpu--load--1min--dataset \
--estimator tf.estimator.DNNClassifier \
--hidden-layers 100/100/100/100/100 \
--steps $(( 2000 / 128 * 500 )) \
--batch-size 128 \
--epochs 500 \
--data-path s3://cimldatasets

```bash
# Train the model based on the dataset and experiment
# Store the evaluation metrics as a JSON file
```
ciml-train--model --dataset cpu--load--1min--dataset \
--experiment dnn--5x100 \
--data-path s3://cimldatasets
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

FfDL Architecture - Source:
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
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>wrt</th>
<th>1m</th>
</tr>
</thead>
<tbody>
<tr>
<td>16/03/2018 21:44</td>
<td>6.1</td>
<td>7.36 \cdot 10^6</td>
<td>5.78 \cdot 10^6</td>
<td>0.97</td>
</tr>
<tr>
<td>16/03/2018 21:44</td>
<td>7.45</td>
<td>7.43 \cdot 10^6</td>
<td>3.6 \cdot 10^5</td>
<td>0.97</td>
</tr>
<tr>
<td>16/03/2018 21:44</td>
<td>4.27</td>
<td>7.31 \cdot 10^8</td>
<td>4.01 \cdot 10^5</td>
<td>0.97</td>
</tr>
<tr>
<td>16/03/2018 21:44</td>
<td>1</td>
<td>7.43 \cdot 10^8</td>
<td>4.096</td>
<td>0.97</td>
</tr>
<tr>
<td>16/03/2018 21:44</td>
<td>0.5</td>
<td>7.44 \cdot 10^8</td>
<td>1.5 \cdot 10^7</td>
<td>0.97</td>
</tr>
<tr>
<td>16/03/2018 21:44</td>
<td>1.75</td>
<td>7.31 \cdot 10^8</td>
<td>4.096</td>
<td>0.97</td>
</tr>
<tr>
<td>16/03/2018 21:44</td>
<td>0.88</td>
<td>7.43 \cdot 10^8</td>
<td>4.096</td>
<td>0.9</td>
</tr>
<tr>
<td>16/03/2018 21:44</td>
<td>1.39</td>
<td>7.31 \cdot 10^8</td>
<td>4.51 \cdot 10^5</td>
<td>0.9</td>
</tr>
<tr>
<td>16/03/2018 21:45</td>
<td>1.01</td>
<td>7.44 \cdot 10^8</td>
<td>4.096</td>
<td>0.9</td>
</tr>
<tr>
<td>16/03/2018 21:45</td>
<td>0.75</td>
<td>7.46 \cdot 10^8</td>
<td>61440</td>
<td>0.9</td>
</tr>
<tr>
<td>16/03/2018 21:45</td>
<td>1.26</td>
<td>7.31 \cdot 10^8</td>
<td>4.096</td>
<td>0.9</td>
</tr>
<tr>
<td>16/03/2018 21:45</td>
<td>1.13</td>
<td>7.44 \cdot 10^8</td>
<td>4.096</td>
<td>0.82</td>
</tr>
<tr>
<td>16/03/2018 21:45</td>
<td>5.77</td>
<td>7.77 \cdot 10^8</td>
<td>1.72 \cdot 10^5</td>
<td>0.82</td>
</tr>
<tr>
<td>16/03/2018 21:45</td>
<td>9.85</td>
<td>8.31 \cdot 10^8</td>
<td>4.99 \cdot 10^6</td>
<td>0.82</td>
</tr>
<tr>
<td>16/03/2018 21:45</td>
<td>3.88</td>
<td>8.46 \cdot 10^8</td>
<td>8.25 \cdot 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**
DNN - Binary Classification

- Classes: Passed or Failed
- Supervised training
- TensorFlow `DNNClassifier`, classes=2

Dataset:
- CI Job "tempest-full"
- Gate pipeline only
- 2000 examples, 1400 training, 600 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}, \texttt{1m}) tuple
- Accuracy achieved: 0.995
- 3 mistakes on a 600 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: 1min
- Accuracy achieved: 0.995
- 3 mistakes on a 600 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

<table>
<thead>
<tr>
<th>metric</th>
<th>tempest-full</th>
<th>tempest-full-py3</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy</td>
<td>0.997</td>
<td>0.943</td>
</tr>
<tr>
<td>loss</td>
<td>65.497</td>
<td>242.369</td>
</tr>
<tr>
<td>auc_precision_recall</td>
<td>0.965</td>
<td>0.608</td>
</tr>
</tbody>
</table>
Binary Classification - Summary

- User CPU and 1min Load Avg
- Resolution: 1 minute is enough
- High accuracy: **0.995**
- 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=9
- Dataset:
  - CI Job "tempest-full"
  - Gate pipeline only
  - 2000 examples, 1400 training, 600 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

Network Graph - Source: TensorBoard
DNN - Multi Class

- Features: (usr, 1m)
- Resolution: 1min
- Loss converges, but...
- Evaluation accuracy achieved: \textbf{0.668}
- 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: 0.668
- 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: 0.668
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: (usr, 1m)
  - Resolution: 1min
  - 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

- Summary on DNN binary classification
- Summary on DNN multi class
- Collect data
- Know your data
- Work with cloud tools
Future Work

- Complete setup of the pipeline
- Human curated dataset for supervised training
- Making our life easier
- Integrate with real life CI system
- Explore job portability
References

- This talk: https://github.com/afrittoli/ciml_talk
- CIML: https://github.com/mtreinish/ciml
Thank you!
Questions?
Natural disasters are among the world’s greatest challenges.

How would 22 million developers solve these global issues if given a chance to answer the call?

Find out how at developer.ibm.com/callforcode
Start solving the world's largest natural disaster challenges through code

1. REGISTER FOR THE CHALLENGE
   callforcode.org/oss
   Learn about the competition scope, prizes, rules and schedule

2. SIGN UP FOR AN IBM ACCOUNT
   ibm.biz/engagewithdeveloper
   Get an account granting free tier access for an unlimited time

3. START BUILDING WITH CODE
   developer.ibm.com/callforcode
   Use code patterns in six tech areas to inspire your submission

4. ENGAGE THE COMMUNITY
   Through the IBM Coder community and Slack
   Meet advocates, brainstorm ideas, find a team, hear about updates

5. SUBMIT YOUR SOLUTION
   ibm.biz/callforcodesubmit
   Describe your app, name your team, link to your code and demo