Inhalt

JOSEPH PARETI JOEPARETI54@GMAIL.COM SUNDAY, JANUARY 28, 2018 1
Machine Learning experiment in the Azure cloud 1
Local setup 1
AZURE VM 2
Model solution 6
Appendix: run Caffe/LeNet on the Azure VM 7

JOSEPH PARETI JOEPARETI54@GMAIL.COM SUNDAY, JANUARY 28, 2018

Machine Learning experiment in the Azure cloud

This report is about setting up a VM in the Azure cloud to be used to run a machine learning experiment for predictive maintenance. The overall system has 2 main components:

1. A workbench software tool that runs local on my laptop
2. A VM in the Azure cloud that communicates with the local workbench over internet

For the VM, a free Azure subscription has been used (which comes with no support from Microsoft), however to speed up problems resolution, a paid subscription with attached support is more adequate.

Because the application at hand is deployed in a docker container that comes preinstalled in the VM, the exercise appears to be a valid 'step 0' in view of defining a suitable use case for Ubercloud, which also makes use of container technology.

One must also understand if the code in this exercise is generic enough to be applied to different predictive maintenance scenarios, so that other users could just ingest their data in the model, or if (non trivial) code modifications are needed.

Finally, quite a lot of work has been invested (by me) to make this project run in the VM, however there are still some stumbling blocks that could be due to:

- Wrong configurations, wrong usage, inadequate software settings or VM template, etc.
- Adequate internet connection (currently, ADSL delivering 20 Mbit maximum)
- Untested software (more unlikely, since Microsoft and github have published the results)

As a result of those issues, at the time of writing this report I could not verify the entire project by myself, and hence I am relying on Microsoft and github publications.

Local setup

- Intel laptop running Windows 10 Home Edition (64 bit)
- ADSL internet connection delivering 20 Mbit theoretical maximum bandwidth

- Azure Machine Learning workbench installed from AmlWorkbenchSetup.msi (version 0.1.1711.15323)

AZURE VM

Free subscription.
Hostname ubuntu-4-ML-new with ip address 52.166.236.107
VM class standard A2m_v2 with 2 vcpus and 16 GB memory

The specifications in:

https://docs.microsoft.com/en-us/azure/machine-learning/preview/scenario-deep-learning-for-predictive-maintenance

call for DS4_V2. The choice of A2m_v2 was dictated to comply with the free subscription terms. The following link provides some benchmark data to characterize DS VMs vs. D*v2 VMs, basically stating that D* delivers better performance than DS, however it offers no comparison with A* VMs:

https://cloudspectator.com/microsoft-azure-dv2-vs-ds-comparison/

Therefore one could ask whether the choice of an A* VM template is appropriate for the task at hand, and whether the trade-off (of an 'A' vs. 'D' type VM) could explain the run time errors reported in the next paragraph.

SOFTWARE STACK

Upon creating the vm, the software stack is automatically available as per:

https://docs.microsoft.com/de-de/azure/machine-learning/data-science-virtual-machine/dsvm-ubuntu-intro

In addition, the following was set:

resource group jp-resource-group
username
speicherkonto jpstorage5a

BASIC TESTS

The purpose of running basic ML tests was to determine whether the software stack in the VM delivers what is supposed to deliver: among various components installed a promising one is caffe because it comes with a sample network.

I used the following guidelines to test the LeNet using caffe:
  http://caffe.berkeleyvision.org/gathered/examples/imagenet.html

I found that not all required components are available, and hence I tried to fill in the gaps by downloading the missing ones, however the caffe setup became unstable and a core was dumped. See appendix for usage details.

My understanding is that this case should run ootb.

SETUP THE ENVIRONMENNT FOR THE ML PREDICTIVE MAINTENANCE WORKSHOP

The ootb stack was not complete, namely:

(i) reset the password as a workaround for an initial problem that prevented "ssh joepareti54@ip-address"
(ii) edit the /etc/sudoers file and append the line
joepareti54 ALL=(ALL) NOPASSWD: ALL

The file modification was needed or else execution would stop with the message

"unable to run docker with sudo"

The message would appear upon executing the following PowerShell command from local workbench:

> az ml experiment prepare -c bar

In addition, the description at:

https://docs.microsoft.com/en-us/azure/machine-learning/preview/scenario-deep-learning-for-predictive-maintenance

is incomplete and the description in the workbench should be used instead(the difference is that you must edit the username and key in the code).

PROJECT SETUP IN LOCAL WORKBENCH

---

setup based on GENERAL PREDICTIVE MAINTENANCE
project name = jp-project-5
resource group = jp-resource-group

joepareti@LAPTOP-4UPCRKBJ /cygdrive/c/Users/joepareti/Documents/jp-project-5/aml
_config
$ cat fooZZZ.compute
address: 52.166.236.107
baseDockerImage: microsoft/mmlspark:plus-0.9.9
nativeSharedDirectory: ~/.azureml/share/
password:
AzureMlSecret=fooZZZ#joepareti54#c07907cad53e439fb0e278f1c4cdbb5b
sharedVolumes: true
type: remotedocker
username: joepareti54

joepareti@LAPTOP-4UPCRKBJ /cygdrive/c/Users/joepareti/Documents/jp-project-5/aml
_config

```
$ cat fooZZZ.runconfig
ArgumentVector:
- $file
CondaDependenciesFilye: aml_config/conda_dependencies.yml
EnvironmentVariables: null
Framework: PySpark
PrepareEnvironment: false
SparkDependenciesFile: aml_config/spark_dependencies.yml
Target: fooZZZ
TrackedRun: true
```

All instructions are explained in the workbench that runs local (in my case it is Windows 10 home edition).

Once the project is set up in local workbench, you can start an interactive session in powershell which then creates a jupyther notebook (using az ml notebook start), so that one can:

1. Verify that the kernel is up and running in trusted state
2. Go ahead with the code in the jupyther browser.

RUN TIME ERRORS in STAGE 2: **"OSError: raw write() returned invalid length"**

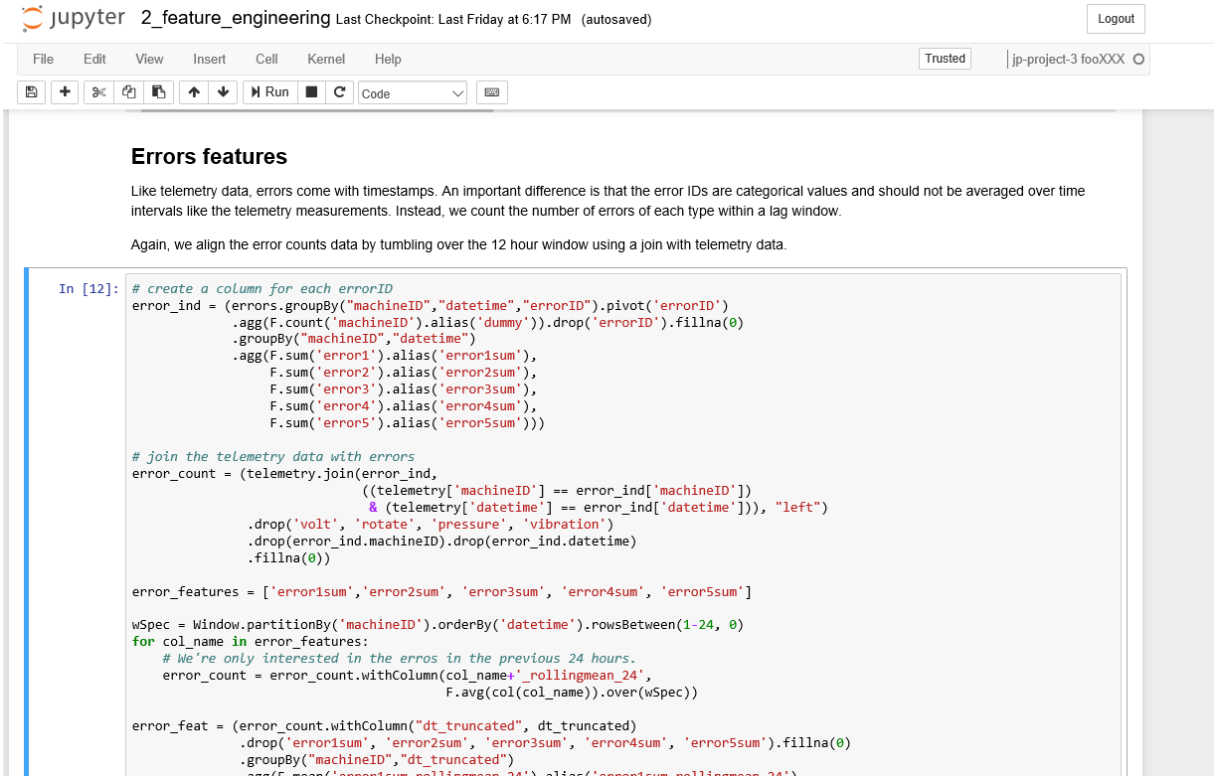
```
[Stage 272:(71 + 2) / 200][Stage 273:==>(2 + 0) / 3][Stage 281:>  (0 + 0) / 3]
[Stage 272:(72 + 2) / 200][Stage 273:==>(2 + 0) / 3][Stage 281:>  (0 + 0) / 3]
{
  "error": {
    "code": "ServiceError",
    "message": "InternalServerError",
    "target": null,
    "details": [],
    "innerError": null,
    "debugInfo": null
  },
  "correlation": {
    "operation": "19980688-4c25fddf697db9c3"
  }
}
[Stage 272:(73 + 2) / 200][Stage 273:==>(2 + 0) / 3][Stage 281:>  (0 + 0) / 3]
Traceback (most recent call last):
  File "C:\Users\joepareti\AppData\local\AmlWorkbench\Python\lib\runpy.py", line 184, in
_run_module_as_main
    "__main__", mod_spec)
  File "C:\Users\joepareti\AppData\local\AmlWorkbench\Python\lib\runpy.py", line 85, in
_run_code
    exec(code, run_globals)
  File "C:\Users\joepareti\AppData\local\AmlWorkbench\Python\lib\site-
packages\azureml\notebooks\_scripts\kernel_launcher.py", line 223, in <module>
    job_name=job_name)
  File "C:\Users\joepareti\AppData\local\AmlWorkbench\Python\lib\site-
packages\azureml\execution\commands.py", line 81, in start
    prepare_environment)
  File "C:\Users\joepareti\AppData\local\AmlWorkbench\Python\lib\site-
packages\azureml\execution\commands.py", line 209, in _start_internal
    async, wait, status["runId"], service_context)
  File "C:\Users\joepareti\AppData\local\AmlWorkbench\Python\lib\site-
packages\azureml\execution\commands.py", line 257, in print_details
    printed = incremental_print(status["driverLog"], printed)
  File "C:\Users\joepareti\AppData\local\AmlWorkbench\Python\lib\site-
packages\azureml\execution\commands.py", line 225, in incremental_print
    print(line)
OSError: raw write() returned invalid length 160 (should have been between 0 and 80)
```

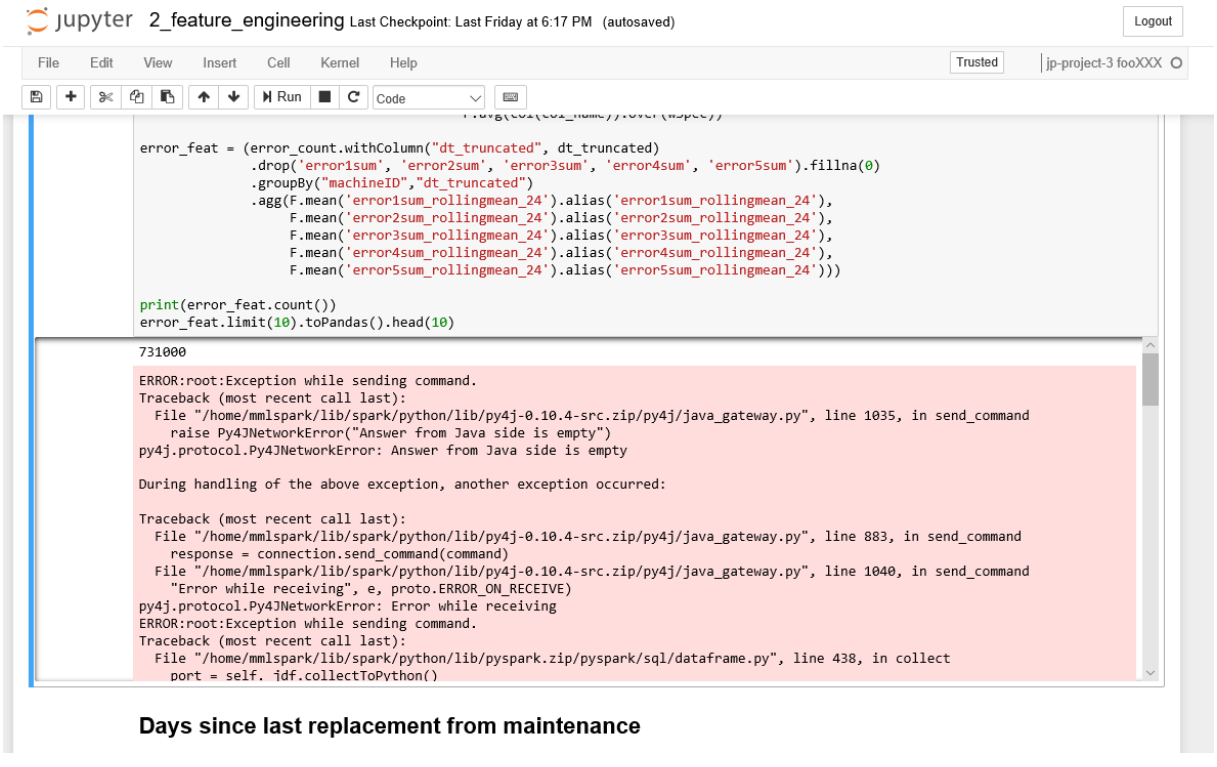
ASSUMPTION:
The above OS error could be due to *a client disconnecting early in the request and server cannot read data*, as suggested in:

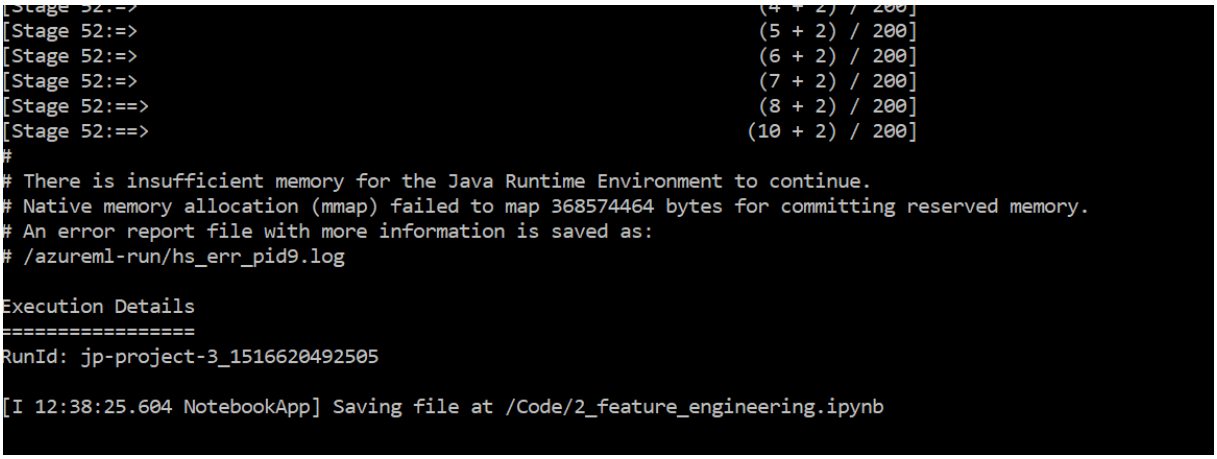
https://github.com/nameko/nameko/issues/368

RUN TIME ERRORS in STAGE 2: "Answer from JAVA stack is empty", or "insufficient memory for JAVA"?

The errors shown below appear in the jupyter window and in the powershell console. However, I could not reproduce them in a subsequent run.







Assumption:

There seems to be a correlation with *Spark container in broken state causing a* ***connection refused***:

https://issues.apache.org/jira/browse/SPARK-18523

RUN TIME ERRORS in STAGE 2: "ConnectionRefused"

```
[Stage 118:(190 + 2) / 200][Stage 120:>(0 + 0) / 200][Stage 122:>  (0 + 0) / 2]
[Stage 118:(198 + 2) / 200][Stage 120:>(0 + 0) / 200][Stage 122:>  (0 + 0) / 2]

[I 11:39:01.265 NotebookApp] Saving file at /Code/2_feature_engineering.ipynb
Traceback (most recent call last):
  File "C:\Users\joepareti\AppData\local\AmlWorkbench\Python\lib\site-
packages\requests\packages\urllib3\connection.py", line 142, in _new_conn
    (self.host, self.port), self.timeout, **extra_kw)
  File "C:\Users\joepareti\AppData\local\AmlWorkbench\Python\lib\site-
packages\requests\packages\urllib3\util\connection.py", line 98, in create_connection
    raise err
  File "C:\Users\joepareti\AppData\local\AmlWorkbench\Python\lib\site-
packages\requests\packages\urllib3\util\connection.py", line 88, in create_connection
    sock.connect(sa)
ConnectionRefusedError: [WinError 10061] Es konnte keine Verbindung hergestellt werden, da der
Zielcomputer die Verbindung verweigerte
```

Model solution

The following website has a description of this ML application including the output of the single code cells for all stages in the project:

https://github.com/Azure/MachineLearningSamples-DeepLearningforPredictiveMaintenance

The following paragraph offers some encouraging analysis towards making the model available for predictive maintenance in real world cases:

The traditional predictive maintenance machine learning models are based on feature engineering which is manual construction of right features using domain expertise and similar methods. This usually makes these models hard to reuse since feature engineering is specific to the problem scenario and the available data which varies from one business to the other. Perhaps the most attractive part of applying deep learning in the predictive maintenance domain is the fact that these networks can automatically extract the right features from the data, eliminating the need for manual feature engineering.

When using LSTMs in the time-series domain, one important parameter to pick is the sequence length which is the window for LSTMs to look back. This may be viewed as similar to picking window_size = 5 cycles for calculating the rolling features in the Predictive Maintenance Template which are rolling mean and rolling standard deviation for 21 sensor values. The idea of using LSTMs is to let the model extract abstract features out of the sequence of sensor values in the window rather than engineering those manually. The expectation is that if there is a pattern in these sensor values within the window prior to failure, the pattern should be encoded by the LSTM.

One critical advantage of LSTMs is their ability to remember from long-term sequences (window sizes) which is hard to achieve by traditional feature engineering. For example, computing rolling averages over a window size of 50 cycles may lead to loss of information due to smoothing and abstracting of values over such a long period, instead, using all 50 values as input may provide better results.

Appendix: run Caffe/LeNet on the Azure VM

This paragraph provides details on running Caffe in the VM, using LeNet as described in the main report. It appears that something is either misconfigured or missing in the software environment.

```
cd $CAFFE_ROOT
sudo wget https://sourceforge.net/projects/libpng/files/zlib/1.2.9/zlib-1.2.9.tar.gz/download
mv download zlib-1.2.9.tar.gz
sudo mv download zlib-1.2.9.tar.gz
sudo gunzip zlib-1.2.9.tar.gz
sudo tar xvf zlib-1.2.9.tar
cd zlib-1.2.9/
./configure
sudo ./configure
sudo make
sudo make install
ls libz.so.1
sudo ln -s -f /usr/local/lib/libz.so.1.2.9/lib libz.so.1
cd ..
./data/mnist/get_mnist.sh
sudo ./examples/mnist/create_mnist.sh
$CAFFE_ROOT
./examples/mnist/train_lenet.sh


./build/tools/caffe: /anaconda/lib/libtiff.so.5: no version information available (required by
/usr/lib/x86_64-linux-gnu/libopencv_highgui.so.2.4)
I0128 16:16:00.956418 28604 caffe.cpp:218] Using GPUs 0
I0128 16:16:00.993460 28604 caffe.cpp:223] GPU 0: fï¿½R
F0128 16:16:00.993695 28604 common.cpp:152] Check failed: error == cudaSuccess (30 vs. 0)
unknown error
*** Check failure stack trace: ***
    @     0x7ff97b2e25cd  google::LogMessage::Fail()
    @     0x7ff97b2e4433  google::LogMessage::SendToLog()
    @     0x7ff97b2e215b  google::LogMessage::Flush()
    @     0x7ff97b2e4e1e  google::LogMessageFatal::~LogMessageFatal()
    @     0x7ff97b87cdb2  caffe::Caffe::SetDevice()
    @           0x40c328  train()
    @           0x408420  main
    @     0x7ff97a1f0830  __libc_start_main
    @           0x408c49  _start
    @              (nil)  (unknown)
./examples/mnist/train_lenet.sh: line 4: 28604 Aborted                 (core dumped)
./build/tools/caffe train --solver=examples/mnist/lenet_solver.prototxt $@
```