The Smart Maintenance Demo

by Joe Hahn, joe.hahn@infochimps.com, 20 May 2015

This is the Github repository for the master branch of the Smart Maintenance Demo for Hadoop. This demo uses the Support Vector Machines (SVM) algorithm to perform predictive maintenance on 200 simulated motors, with most (but not all) of the computation being done in parallell on the Hadoop cluster's datanodes via Spark.

To install:

First clone this github repo to your home directory on the hadoop foyer node:

```
cd; git clone git@github.com:infochimps-sales/spark-airline-demo.git
```

Then execute the installer, this will download and install some python libraries to all hadoop nodes, and is done in 5 minutes:

```
cd spark-airline-demo
./install.sh
```

To execute:

To submit this spark job to Yarn for execution:

```
PYSPARK_PYTHON=/home/$USER/anaconda/bin/python spark-submit smart_maint_spark.py
```

Monitor this job's progress using the Spark UI by browsing:

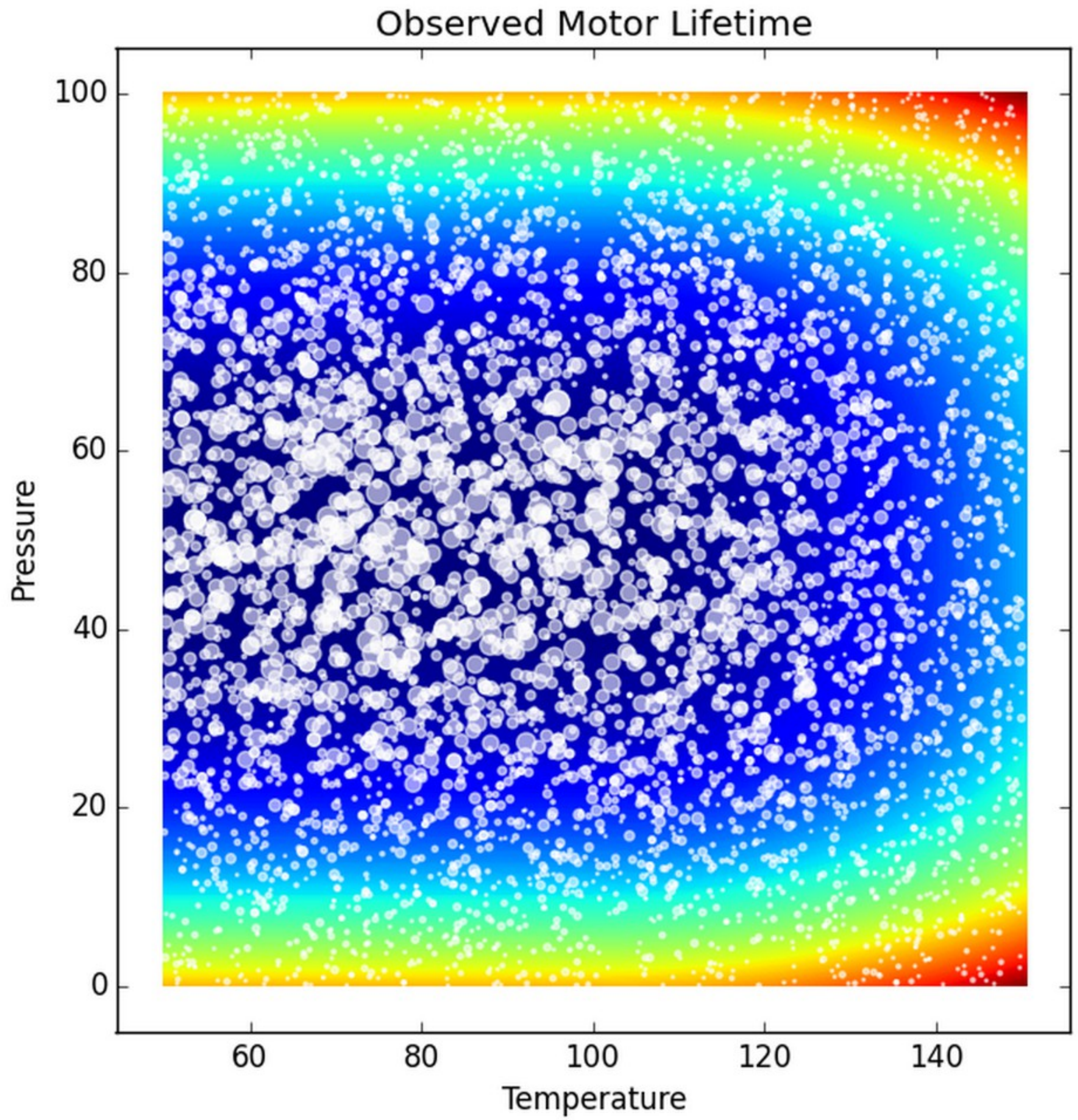
```
Cloudera Manager -> Home -> Yarn -> Resource Manager UI -> application_ID# -> Applicati
```

The output of this spark job is 3 png images that can be viewed by browsing

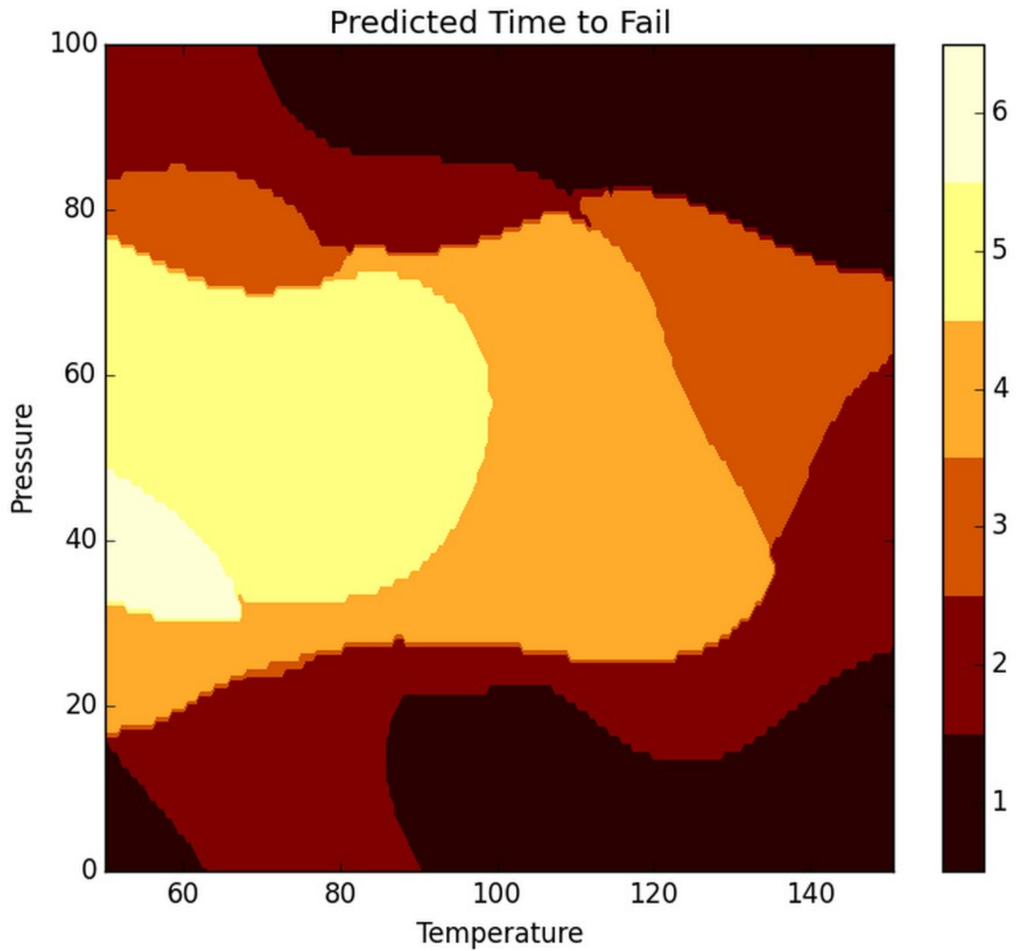
```
http://cdh-foyer.platform.infochimps:12321/figs
```

The demo's storyline:

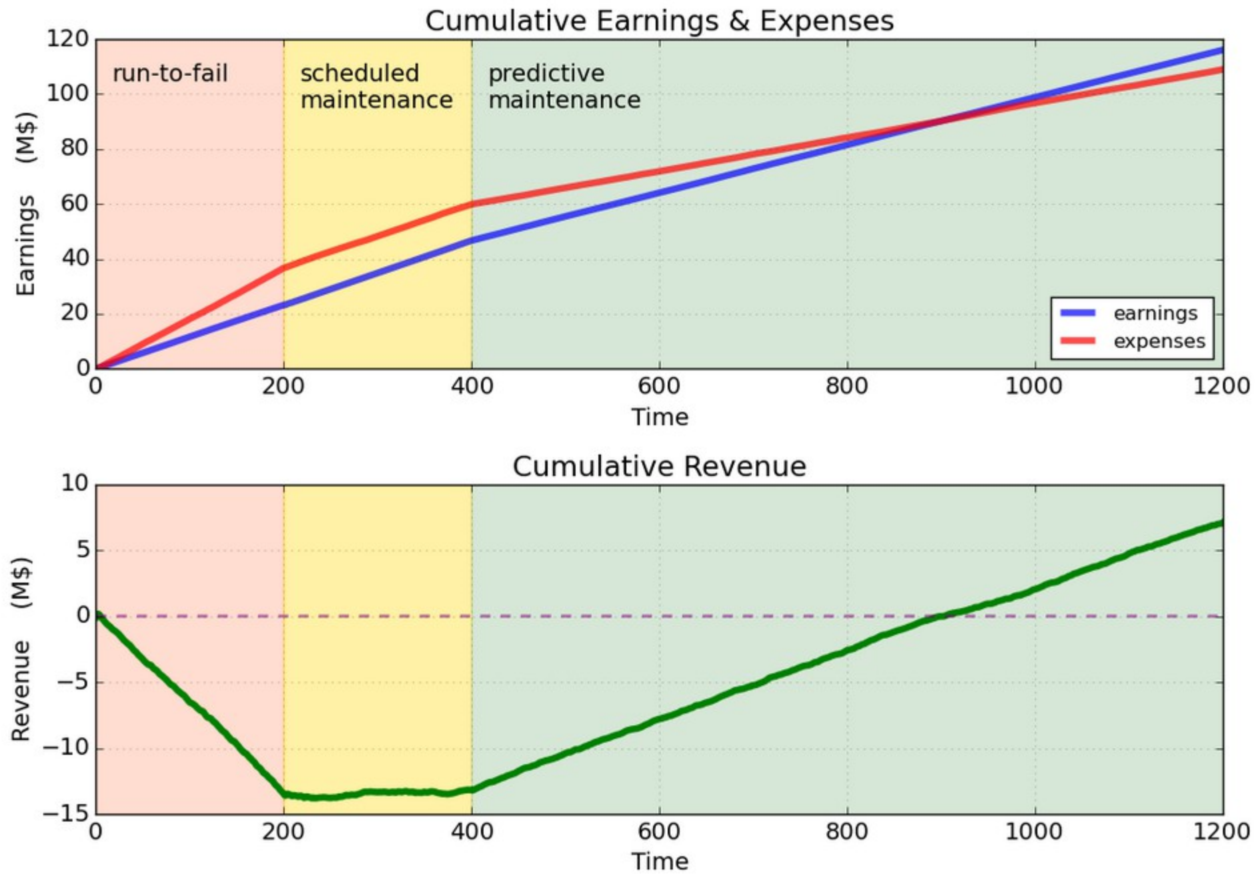
This demo calculates the operational history of 200 simulated motors over time. Initially these motors are evolved using a *run-to-fail* maintenance strategy. Each motor has two knobs, Pressure (P) and Temperature (T), and the size of the dots in the following scatterplot shows that the longest-lived motors have (P,T) settings in these intervals: 40 < P < 60 and T < 100, with motors being progressively shorter-lived the further their P,T setting are from the sweet spot at P~50 and T<100:



The demo evolves these motors in run-to-fail mode until time t=200, and then (just for kicks) it switches to a *scheduled-maintenance* strategy during times 200 < t < 400. During scheduled-maintenance operation, every engine is sent to maintenance every 5 days, this simply removes some cruft and temporarily reduces the likelihood of motor failure. Meanwhile the SVM algorithm is trained on the run-to-fail data, which is simply the observed engine lifetimes versus their (P,T) settings. Once trained, the SVM algorithm is now able to use an engine's (P,T) settings to predict that engine's lifetime ie its estimated time-to-fail. Thereafter (at times t > 400) the engines are evolved using *predictive-maintenance*, which simply sends an engine into maintenance when its predicted time-to-fail is one day hence. The following diagram shows the SVM's so-called *prediction surface*, which map's the engines' predicted time-to-fail across the (P,T) parameter space. Note that SVM's predicted time-to-fail does indeed recover the engines' sweet-spot at 40 < P < 60 and T < 100, though the edges of the predicted stable zone is somewhat ragged.



Each operating engine also generate earnings at a rate of $1000/day, while engines that are being maintained instead generate modest expenses (-$200/day), with failed engines generating larger expenses (-$2000/day) while they are in the shop for repairs. The following shows that operating these engines in *run-to-fail* mode is very expensive, resulting in cumulative earnings of -$13M by time t=200. This plot also shows that operating these engines using a *scheduled-maintenance* strategy is a wash, with earnings nearly balancing expenses. But switching to a *predictive-maintenance* strategy at t=400 then results in earnings that exceeds expenses, so much so that the operators of these engines recover all lost earnings by time t=850, and have earned $7M at the end of this simulation.



So this demo's main punchline is: *get Smart Maintenance on the BDPaas to dramatically reduce expenses and to grow earnings.*

Known issues:

1 If the png images are not browse-able, restart the webserver on the hadoop foyer node:

```
/home/$USER/anaconda/bin/python -m SimpleHTTPServer 12321 > /dev/null 2>&1 &
```

2 Close inspection of Spark's Application Master UI will show that this job is being executed on only 2 of the 3 available datanodes, I have no idea why one datanode is not participating, this needs to be debugged.

3 Spark's console output is *way* too verbose, I attempted to dial that down on foyer node via:

```
sudo cp /opt/cloudera/parcels/CDH-5.3.0-1.cdh5.3.0.p0.30/etc/spark/conf.dist/log4j.prop
    /opt/cloudera/parcels/CDH-5.3.0-1.cdh5.3.0.p0.30/etc/spark/conf.dist/log4j.properti
```

and in log4j.properties set

```
log4j.rootCategory=WARN, console
```

but the above didn't help any...perhaps one must do this on all datanodes?

Debugging notes:

One can execute this demo line-by-line at the python command line using pyspark, this is useful when debugging code:

```
PYSPARK_PYTHON=/home/$USER/anaconda/bin/python pyspark
```

Then copy-n-past each line from smart_maint_spark.py into the python command line, EXCEPT for line 25: sc = SparkContext(conf=conf...

To get pyspark to use ipython (rather than python):

```
sudo rm -f /usr/bin/python /usr/bin/ipython
sudo ln -s /home/$USER/anaconda/bin/python /usr/bin/python
sudo ln -s /home/$USER/anaconda/bin/ipython /usr/bin/ipython
IPYTHON=1 pyspark
```

And to undo the above changes:

```
sudo rm /usr/bin/python /usr/bin/ipython
sudo ln -s /usr/bin/python2.6 /usr/bin/python
```