README.md

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 of the computations being done in parallell across the Hadoop cluster's datanodes using Spark. Python's Bokeh library is also used to generate an interactive dashboard that visualizes the results of this simulation.

To install:

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

```
cd; git clone git@github.com:infochimps-sales/smart-maintenance-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 smart-maintenance-demo
./install.sh
```

To execute:

To submit this spark job to Yarn for execution:

```
PYSPARK_PYTHON=/home/$USER/anaconda/bin/python spark-submit \
    --master yarn-client --num-executors 3 --executor-cores 6 --executor-memory 4G \
    --driver-java-options "-Dlog4j.configuration=file:///home/$USER/smart-maintenance-demo/log4j
    smart_maint.py
```

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

```
Cloudera Manager -> Home -> Yarn -> Resource Manager UI -> application_ID# -> Application Maste
```

After the spark job completes, execute this script to dashboard the results of the smart-maintenance simulation,

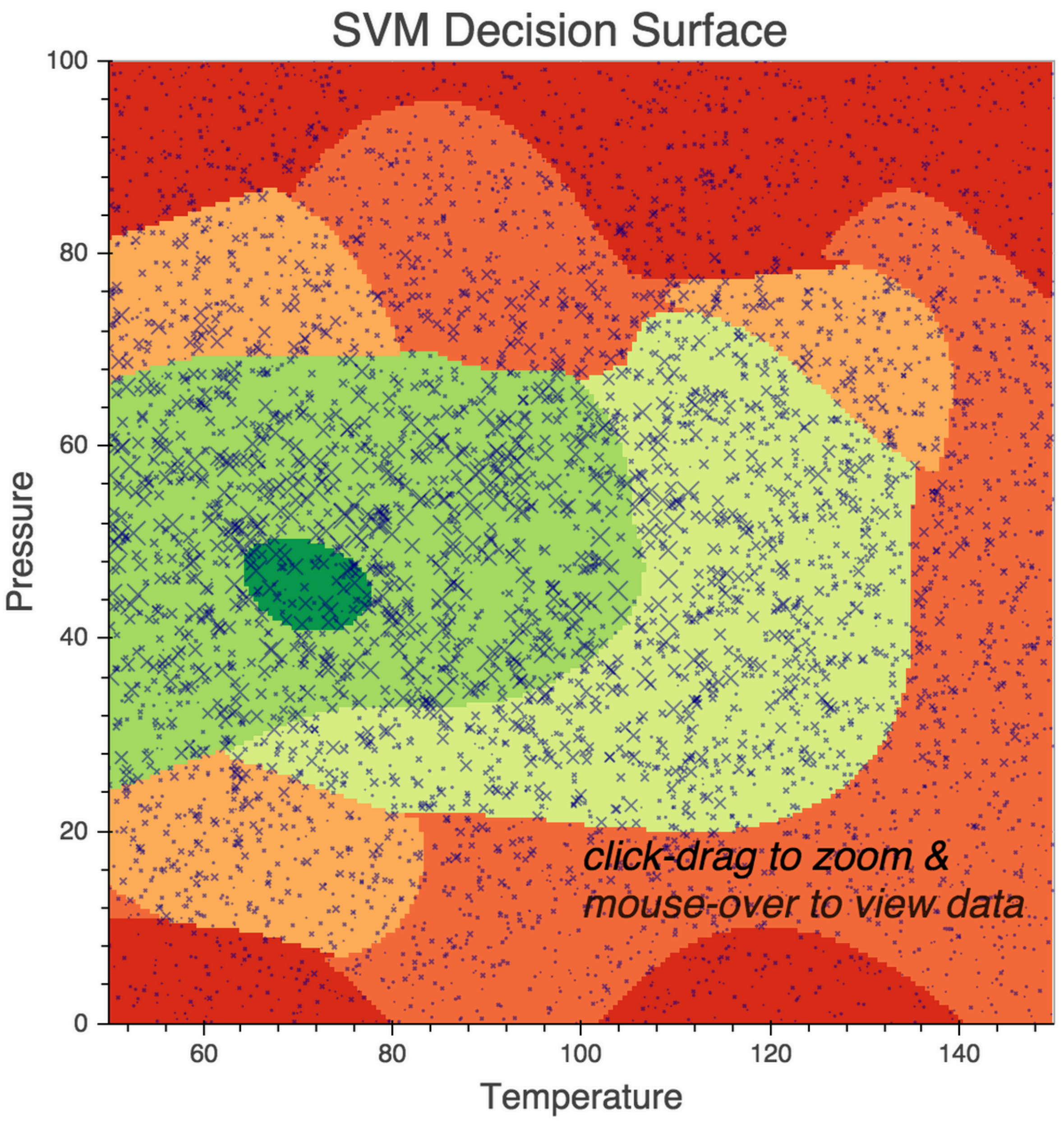
```
/home/$USER/anaconda/bin/python dashboard.py > /dev/null
```

and then browse the resulting dashboard at

```
http://cdh-foyer.platform.infochimps:12321/dashboard.html
```

The demo's storyline:

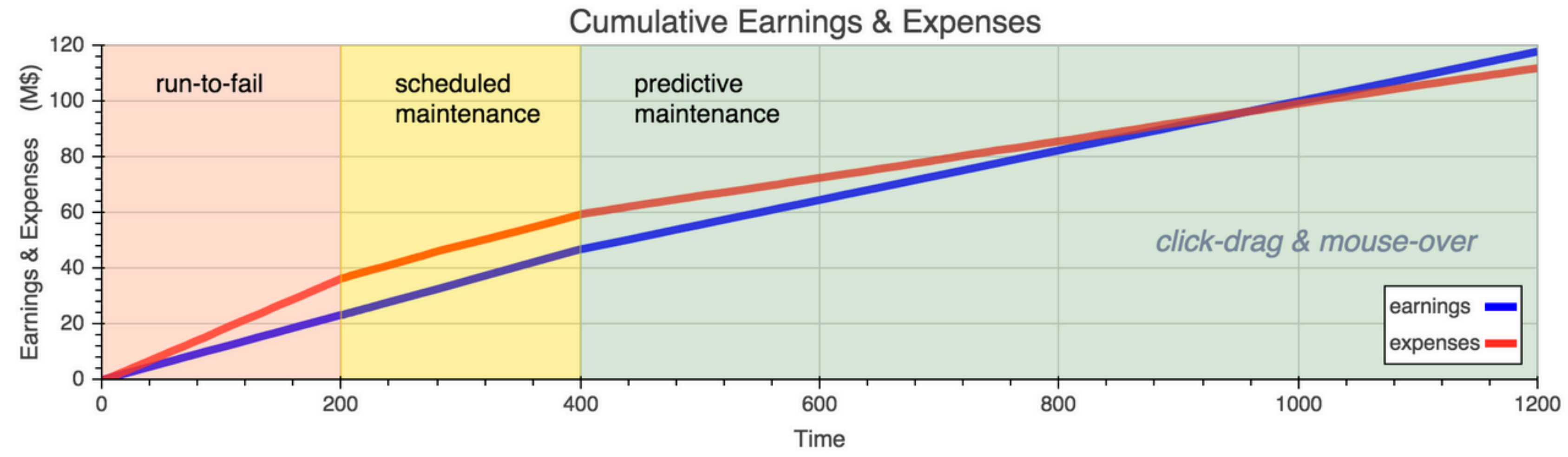
This demo calculates the operational history of 200 simulated motors over time. Initially these motors are evolved using a *run-to-fail* maintenance strategy, and the motor data collected during this period is shown below. Each motor has two knobs, Pressure (P) and Temperature (T), and motors having P,T settings in the interval 40 < P < 60 and T < 100 are by design longest lived, while motors having P,T setting further from the sweet spot at P~50 and T<100 are progressively shorter lived. This trend is also indicated by the size of the crosses in the following scatterplot, which shows how engine lifetime varies with P,T.

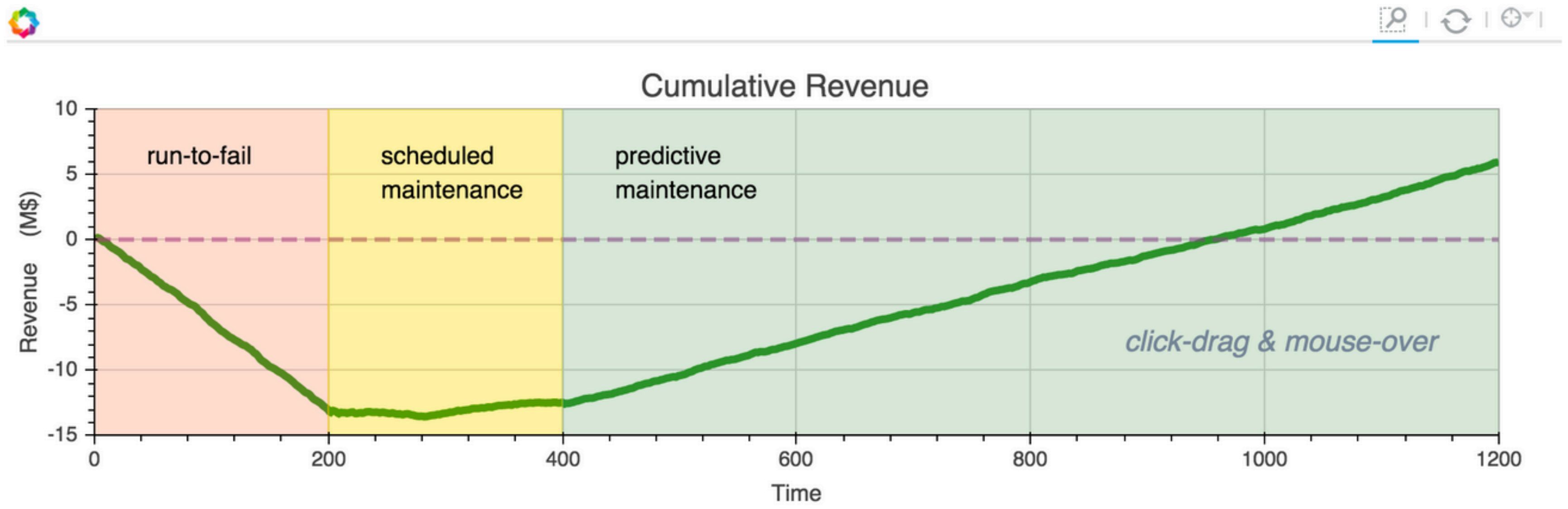


All plots in this demo's dashboard (see http://cdh-foyer.platform.infochimps:12321/dashboard.html) are interactive; click-drag to zoom in on a region, and mouse-over individual data to see their their values.

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 coloring in the above plot also 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 the 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 are rather fluid.

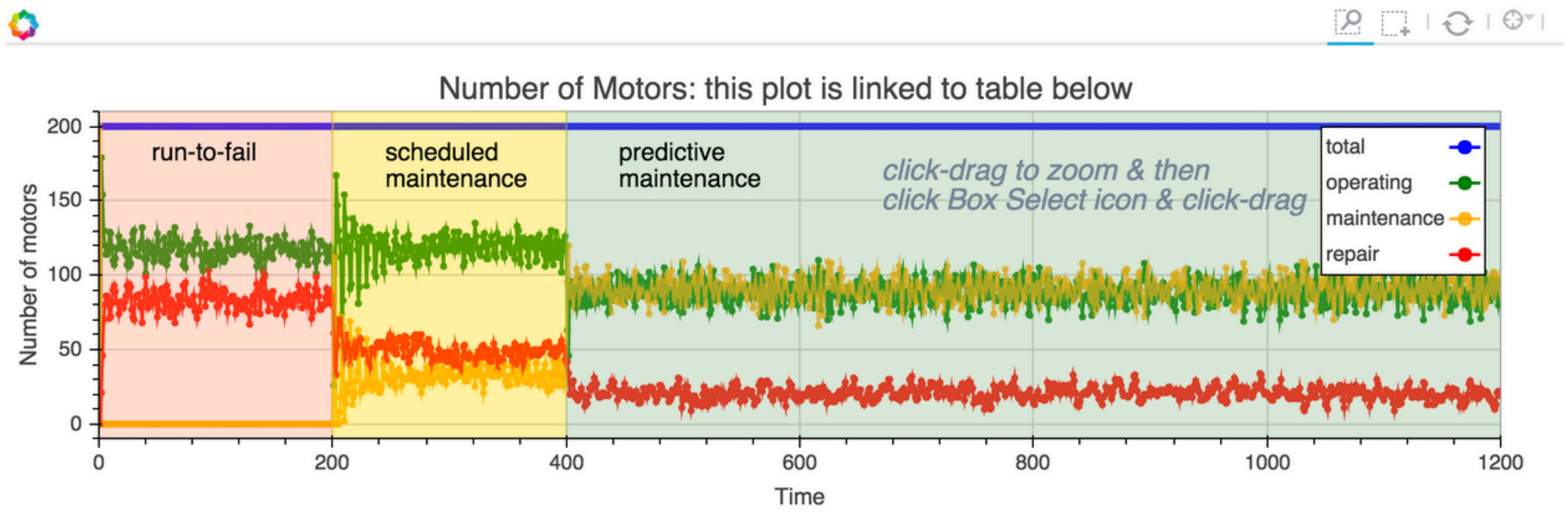
Each operating engine also generate earnings at a rate of $1000/day, while engines that are being maintained instead generate modest expenses (-$200/day), with failed engines generating larger expenses (-$2000/day) while in the shop for repairs. The following plots show that operating these engines in *run-to-fail* mode is very expensive, resulting in cumulative losses 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=870, and have earned $6M at the end of this simulation.





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

The following plot is some additional dashboard-fu. Again, click-drag to zoom in on some region within this plot. Then click the *Box Select* icon in the upper right and click-drag again. This plot is also linked to the table below, and clicking on any column heading there will force the table to update and display only those datapoints that are highlighted in the upper plot.



Time: click to update table	operating	maintenance	repair	total
1	0	200	0	200
2	179	0	21	200
3	154	0	46	200
4	130	0	70	200
5	120	0	80	200
6	114	0	86	200
7	114	0	86	200
8	117	0	83	200
9	129	0	71	200
10	128	0	72	200
11	119	0	81	200

Known issues:

If the dashboard is not visible in the browser, the webserver likely needs to be restarted:

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

Debugging notes:

To benchmark spark's commandline settings:

```
START=$(date +%s); \
PYSPARK_PYTHON=/home/$USER/anaconda/bin/python spark-submit --master yarn-client \
    --num-executors 3 --executor-cores 6 --executor-memory 4G \
    --driver-java-options "-Dlog4j.configuration=file:///home/$USER/smart-maintenance-demo/log4
    smart_maint.py; \
echo "execution time (seconds) = "$(( $(date +%s) - $START ))
```

One can also 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
```