data_exploration_features

January 26, 2017

0.1 Loading data/libs

```
In [1]: import pandas as pd
        import numpy as np
        import calendar

        from bokeh.charts import output_notebook, Scatter, Bar, show, output_file, Line, BoxPlot
        from bokeh.plotting import figure
        from bokeh.layouts import row, column, gridplot

        from ipywidgets import interactive
        from IPython.display import display
        from IPython.utils.py3compat import annotate

        from graph import build_hist, to_relative_time
        from fft import to_fft

        output_notebook()
```

```
In [2]: INPUT="data/device_failure.csv"
        dataset = pd.read_csv(INPUT,index_col=[0,1],parse_dates=[0])
```

features
Per features:
Statistical distribution: - Distribution - Distribution over failures - Distribution over devices - Distribution over failing devices

0.1.1 Temporal distribution

- Average value over time
- Average value over time for failing devices
- Value before failure

0.1.2 Frequency distribution

- DFT / device
- DFT / failing device

```
In [3]: @annotate(attribute=list(dataset.columns[1:]))
        def pick_attribute(attribute):
            return "current attribute=%s" % s
        s = interactive(pick_attribute)
        display(s)
```

```
'current attribute=<ipywidgets.widgets.widget_box.Box object at 0x7fa41134de50>'
```

0.2 building data objects

```
In [4]: attribute = s.children[0].value
        feature_dset = dataset[[attribute,"failure"]]
        failing_points = feature_dset[feature_dset["failure"]>0]
```

```
In [5]: def failure_date(failure):
            data = feature_dset.ix[failure.index]
            dates =data[data["failure"]>0]
            if not dates.empty:
                return dates.iloc[0].name[0]
            else:
                return None

        devices = feature_dset.groupby(level=1).agg(
            {
                "failure":{
                    "failure":np.sum,
                    "failure_date":failure_date},
                attribute : {
                    "min_att":np.min,
                    "max_att":np.max,
                    "mean_att":np.mean,
                    "std_att":np.std
                }})
        devices.columns = devices.columns.droplevel()

        failing_devices = devices[devices["failure"]>0]
        working_devices = devices[devices["failure"]==0]


        working_devices_t = pd.DataFrame({attribute:feature_dset[attribute].unstack().filter(ite
        failing_devices_t = pd.DataFrame({attribute:feature_dset[attribute].unstack().filter(ite
```

```
In [6]: p0 = build_hist(failing_points,attribute,u"%s for failing points" % attribute,color="gre
        p1 = build_hist(feature_dset,attribute)

        h = row(p0,p1)
        show(h)
```

```
In [7]: def per_col(car):
            p0 = build_hist(failing_devices,car,label="%s on failing devices"% car)
            p1 = build_hist(working_devices,car,label="%s on working devices"% car)
            return [p0,p1]

        show(gridplot([ per_col(c) for c in set(c for c in devices.columns if "att" in c ) ]))
```

~~attrbibute 5 : some failing points might be controlled by too much variation in this attribute~~

0.3 Average Value over Time.

```
In [8]: def dev_std(device_df):
            return device_df.rolling(window=20,center=False).std()

        def roll_std(df):
            return df[attribute].groupby(level="device").transform(dev_std)


        failing_devices_t["rolling_std"]  = roll_std(failing_devices_t)
        working_devices_t["rolling_std"]  = roll_std(working_devices_t)

        time = pd.DataFrame({
            attribute : feature_dset[attribute].groupby(level=0).mean(),
            "%s for failing devices" %attribute : failing_devices_t[attribute].groupby(level=1).
            "%s for working devices" %attribute : working_devices_t[attribute].groupby(level=1).
            "rolling std(%s) for failing devices" % attribute : failing_devices_t["rolling_std"]
            "rolling std(%s) for working devices" % attribute : working_devices_t["rolling_std"]
        })
        show(Line(time))
```

attribute1 seems t be completely different at the end of the period:

- ~~Hyp 1 : amplitude gets higher when the device start failing => disproved~~
- ~~Hyp 2 : amplitude is always higher for more fragile devices => disproved~~
- ~~Hyp 2 : wider amplitude for SOME signals~~
- ~~Hyp 3 : failing devices somehow synchronize, resonnance effect (unlikely. plus what would it mean ?)~~
- Hyp 4 : too few devices to average out see graph "n_devices" [here][1] [1]: data_exploration.ipynb

0.3.1 Watching samples aligned on the failure time

```
In [9]: rel_time_threshold = -100
        n_samples = 50

        # relative time is relative to the failure date for negatives
        fail_end_dates= devices["failure_date"].dropna().to_dict()
        fail_relative_time = to_relative_time(failing_devices_t,fail_end_dates,rel_time_threshol
```

```
# for working ones, we use the last value (beware, could lead to weird effects, if the a
work_end_dates= working_devices_t.reset_index(level="date")["date"].groupby(level=0).max
work_relative_time = to_relative_time(working_devices_t,work_end_dates,rel_time_threshol

fail_rel_sampled = fail_relative_time[attribute].unstack(level="dt_from_fail").sample(n=
work_rel_sampled = work_relative_time[attribute].unstack(level="dt_from_fail").sample(n=

show(row(
    Line(
        fail_rel_sampled.unstack(level="device"),
        width=450,
        height=400,
        title ="%s before failure for a sample failing devices" % attribute,
        legend=None),
    Line(
        work_rel_sampled.unstack(level="device"),
        width=450,
        height=400,
        title ="%s before end for a sample working devices" % attribute,
        legend=None)
))
```

attribute 4: Need to take into account the Dv aver time, in addition to the dt

attribute 6 : we can see two classes of population: the ones with increasing attr6, and the ones without

0.3.2 Sampled devices in actual time

In [10]:
```
n_samples=20
sampled_working_devices = working_devices_t[attribute].unstack(level="date").sample(n=n
sampled_failing_devices = failing_devices_t[attribute].unstack(level="date").sample(n=n

l0 = Line(
    sampled_failing_devices.unstack(level="device"),
    width=450,
    height=400,
    title='%s for sampled failing devices' % attribute,
    legend=None
)
l1 = Line(
    sampled_working_devices.unstack(level="device"),
    width=450,
    height=400,
    title = '%s for sampled working devices' % attribute,
    legend=None
)

show(row(l0,l1))
```

0.3.3 FFT

In [11]:
```
fft_df = feature_dset[[attribute]].copy()
fft_per_device = fft_df[attribute].groupby(level="device",sort=True).transform(to_fft)
fft_df["df"] = fft_per_device

fft_plot = fft_df.groupby(level="device").apply(lambda x: x.reset_index(drop=True))["df

fft_working_devices = fft_plot.unstack(level=0).filter(items=working_devices.index).sta
fft_failing_devices = fft_plot.unstack(level=0).filter(items=failing_devices.index).sta

n_samples = 100
to_plot_working = fft_working_devices.unstack(level=1).sample(n=10).unstack().dropna()
to_plot_failing = fft_failing_devices.unstack(level=0).sample(n=10).unstack().dropna()
show(row(
    Line(to_plot_failing.unstack("device"),
        width=450,
        height=400,
        title = "dft of %s for sampled failing devices" % attribute,
        legend=None),
    Line(to_plot_working.unstack("device"),
        width=450,
        height=400,
        title = "dft of %s for sampled working device" % attribute,
        legend=None),
))
```

attribute 6 : kampai !! attribute 9 : good

In [12]:
```
if attribute == u"attribute3":
    dd = feature_dset
    dd[dd[attribute]>0]
    dd = dd.swaplevel().sort_index()
    grouped= dd[dd[attribute]>0].groupby(level="device").agg(lambda x : len(np.unique(x
    weird_devices = set(grouped[grouped>1].index)
    print weird_devices
    weird_values = dd.unstack(level="date").filter(items=weird_devices,axis="index").st
    print weird_values["failure"].value_counts()
    print weird_values.groupby(level="device").apply(lambda df: df[attribute].value_cou
```

In [13]:
```
# attribute 5: testing if variations are not wider for some failures..
print (failing_devices["max_att"] -failing_devices["min_att"]).value_counts()
print (working_devices["max_att"] -working_devices["min_att"]).value_counts()

# answer: Nope. ...
```

```
0       62
8        2
2336     1
```

```
9264      1
160       1
328       1
10288     1
168       1
32        1
88        1
64712     1
368       1
448       1
648       1
62488     1
16        1
63504     1
2576      1
1208      1
520       1
1656      1
21816     1
920       1
48        1
80        1
2168      1
61088     1
240       1
10336     1
27856     1
4360      1
2792      1
2144      1
136       1
2496      1
10200     1
304       1
2408      1
21200     1
976       1
64        1
10440     1
456       1
960       1
dtype: int64
0        1003
8           8
16          5
296         3
40          3
96          2
```

```
144          2
464          2
32           2
24           2
112          1
120          1
416          1
55464        1
176          1
200          1
208          1
8544         1
49192        1
128          1
3976         1
1872         1
912          1
1752         1
3696         1
1384         1
64792        1
1240         1
64584        1
1000         1
2864         1
552          1
51976        1
4832         1
6856         1
2688         1
632          1
616          1
2648         1
424          1
dtype: int64
```

In [14]:
```
## Isolate a validation dataset

# easier without index
all_devices = dataset.reset_index()[["device","failure"]]

#Keep 5% of each class for validating
validation_neg = set(all_devices[all_devices["failure"]==1].sample(frac=0.05, random_st
validation_pos = set(all_devices[all_devices["failure"]==0].sample(frac=0.05, random_st
validation_devices = validation_neg.union(validation_pos)
```

```
# build the two sets
per_device = dataset.reset_index()
per_device["k"] = per_device["device"].map(dict( (k,True) for k in validation_devices))
per_device["k"].fillna(False,inplace=True)
g = per_device.groupby(by="k")
training_set = g.get_group(True).set_index(drop=True,keys=["date","device"])
validation_set = g.get_group(False).set_index(drop=True,keys=["date","device"])
del training_set["k"]
del validation_set["k"]

# save the two sets
print validation_set.shape
print training_set.shape
validation_set.to_csv("data/validation.csv")
training_set.to_csv("data/train.csv")
```

```
(2729, 10)
(121765, 10)
```

```
In [ ]:
```