Model Validation

January 26, 2017

In [1]:
```
import pandas as pd
import numpy as np
import calendar
from sklearn.externals import joblib
import pickle
from sklearn.metrics import roc_curve, accuracy_score,precision_recall_curve, auc

from bokeh.charts import output_notebook, Scatter, Bar, show, output_file, Line, BoxPlot
from bokeh.plotting import figure

from ML import filter_devices, build_deriv, resample_per_device, subsample_negatives
from fft import fft_peak
output_notebook()
```

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

In [3]:
```
#
# apply the feature pre_transformations
#

def pre_filter(df):
    res = df.copy()
    del res["attribute1"]
    #del res["attribute3"]
    #del res["attribute5"]
    dt_list = ["attribute2","attribute8"]
    for c in dt_list:
        deriv = build_deriv(res,c)
        res["dt_%s" % c] = deriv
        res["dt2_%s" % c] =  build_deriv(res,c,2)
    return res.fillna(0)

def post_filter(df):
    res = df.copy()
    res = filter_devices(res)
    for col in res.columns:
        if "min" in col:
```

```
            del res[col]
        if "std" in col:
            del res[col]
    return res
pre_dataset = pre_filter(dataset)
#print feature_set.columns

features = [f for f in pre_dataset.columns if "att" in f]
def f_to_dict(feature):
    indexes = dict( ("avg_over%i_%s" % (i,feature), lambda df: df[:(i+1)].mean()) for i
    d = {
            "min_%s" % feature:np.min,
            "max_%s" % feature:np.max,
            "mean_%s" % feature :np.mean,
            "std_%s" % feature:np.std
        }
    d.update(indexes)
    dft_list = ["attribute4","attribute5", "attribute6","attribute7","attribute9"]
    if feature in dft_list:
        d["dft_p0_ind%s" % feature] = lambda r : fft_peak(r,p=0,index_no_value=True)
        d["dft_p0_val%s" % feature] = lambda r : fft_peak(r,p=0,index_no_value=False)
    return d

agg_dict = dict( (f,f_to_dict(f)) for f in features )

# bugfix: rolling aggregation after group by does not handle multiple aggregation per co
# we fix this by flattening the aggregation dict and repeating the data within the dataf
final_columns = [k for c in agg_dict for k in agg_dict[c]]
input_columns = [c for c in agg_dict for k in agg_dict[c]]
flat_agg_dict = dict( (k,agg_dict[c][k]) for c in agg_dict for k in agg_dict[c])
dup_dataset = pre_dataset[input_columns].sort_index(level="date").sort_index(level="devi
dup_dataset.columns = final_columns

#
# instead of simply grouping by, we roll over the dataset...
# using the "hacky" flat version, to avoid issues.
# The hack increase the memory consumption quite a lot, but it should still be better th
# explicitely building the windowed lines before aggregating over it.
#

feature_set = resample_per_device(dup_dataset) \
    .groupby(level="device",as_index=False) \
    .rolling(window=360,min_periods=1) \
    .agg(flat_agg_dict) \
    .reset_index(level=0,drop=True)
feature_set = post_filter(feature_set).sortlevel(level="device")

#
```

```
        # use label_window to expand label to neighboring days.
        # basically, a mainrtenance x days before failure is still OK
        #
        label_window = 7

        label_set =  resample_per_device(dataset[["failure"]]) \
            .sortlevel(level="date",ascending=False) \
            .groupby(level="device",as_index=False) \
            .rolling(window=label_window, min_periods=1) \
            .sum() \
            .reset_index(level=0,drop=True)
        label_set = filter_devices(label_set).sortlevel(level="device")
```

In [4]:
```
        #
        # load the device model
        #
        model_name = "device"

        device_model = joblib.load("model_%s.pkl" % model_name)
        device_model.named_steps
        with open('model_%s.feat' % model_name, "r") as f:
            device_model_features = pickle.load(f)
```

In [5]:
```
        #
        # Load the device_time model
        #
        model_name = "device_time"

        device_time_model = joblib.load("model_%s.pkl" % model_name)
        device_time_model.named_steps
        with open('model_%s.feat' % model_name, "r") as f:
            device_time_model_features = pickle.load(f)
```

In [9]:
```
        device_mat = feature_set[device_model_features]
        dt_mat = feature_set[device_time_model_features]
        label_mat = label_set.as_matrix()

        device_prediction = device_model.predict_proba(device_mat)[:,1]
        device_time_prediction = device_time_model.predict_proba(dt_mat)[:,1]

        # basic (two models) vote
        confidence_0 = np.abs(device_prediction -0.5)
        confidence_1 = np.abs(device_time_prediction -0.5)
        vote_0 = confidence_0 > confidence_1
        vote_1 =  np.invert(vote_0)

        prediction = vote_0*device_prediction +vote_1*device_time_prediction
        #prediction= device_time_prediction*device_prediction
```

```
In [10]: total_prec, total_recall, ths = precision_recall_curve(label_mat, prediction)
         dev_prec, dev_recall, ths = precision_recall_curve(label_mat, device_prediction)
         dev_t_prec, dev_t_recall, ths = precision_recall_curve(label_mat, device_time_predictio

In [20]: # pr curve
         tpr_f = figure(width=400,height=400,title="PR curve")
         tpr_f.xaxis.axis_label = "recall"
         tpr_f.yaxis.axis_label = "precision"

         tpr_f.cross(dev_recall,dev_prec,size=5,color="green")
         tpr_f.line(dev_recall,dev_prec,legend="device model PR",color="green")

         tpr_f.cross(dev_t_recall,dev_t_prec,size=5,color="red")
         tpr_f.line(dev_t_recall,dev_t_prec,legend="device & time model PR",color="red")

         tpr_f.cross(total_recall,total_prec,size=5)
         tpr_f.line(total_recall,total_prec,legend="combined model PR")

         show(tpr_f)

In [19]: from sklearn.metrics import average_precision_score
         for name , pred in [ ("device model", device_prediction),
             ("device and time model", device_time_prediction),
             ("combined model", prediction)] :
             print "average precision for %s: %.3g" % (name,average_precision_score(label_mat, p

average precision for device model: 0.209
average precision for device and time model: 0.476
average precision for combined model: 0.501
```

The model combination does not perfect ideally ==> try booosting ?

```
In [ ]:
```