

# model\_per\_device

January 26, 2017

```
In [7]: import pandas as pd
import numpy as np
import calendar

from ML import filter_devices, build_deriv, resample_per_device, subsample_negatives
from fft import fft_peak

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 bokeh.io import hplot

from sklearn.model_selection import cross_val_score
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.preprocessing import Normalizer
from sklearn.metrics import roc_curve, auc
from sklearn.svm import SVC, NuSVC
from sklearn.model_selection import LeavePGroupsOut, GroupShuffleSplit
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, accuracy_score, precision_recall_curve, auc
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKfold

output_notebook()

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

## 0.1 per device model

- Set up first model
- Precision/recall, ROC
- Calibration
- PCA ?

- feature engineering
- data cleaning
- Test other models

## 0.2 Build Training set

```
In [3]: 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

In [4]: 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):
    d = {
        "min_%s" % feature: np.min,
        "max_%s" % feature: np.max,
        "mean_%s" % feature : np.mean,
        "std_%s" % feature: np.std
    }
    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 )
        #print agg_dict

In [5]: feature_set = pre_dataset.groupby(level="device").agg(agg_dict)
```

```

feature_set.columns = feature_set.columns.droplevel()
feature_set = post_filter(feature_set)

# feature filtering
# feature filtering
#try :
#    filtered = feature_set.filter(items=feature_imp.index)
#    print "filtering devices"
#    feature_set = filtered
#except:
#    print "no feature filtering"

label_set = dataset[["failure"]].groupby(level="device").sum()
label_set = filter_devices(label_set)
feature_mat = feature_set.as_matrix()
label_mat = label_set.as_matrix().ravel()

```

### 0.3 Run model

```

In [26]: pca = PCA()#n_components="mle",svd_solver="full"
        norm = Normalizer()

#model=GradientBoostingClassifier()
model = RandomForestClassifier()
#model = SVC(probability=True)

pipeline= Pipeline([('normalize', norm),('reduce_dim', pca),("model",model)])

#Fit TPOT result
#pipeline = pipeline_optimizer._fitted_pipeline

try:
    # use best parameters if available
    #
    pipeline.set_params(**grid_result.best_params_)
    print "using last optimized model"
except:
    print "no optim result, or bad ones: let's keep the default ones"
    pass

scores = cross_val_score(pipeline, feature_mat, label_mat,cv=3,verbose=1,scoring="accuracy")
print "accuracy: %g, std(%g)" % (scores.mean(), scores.std())

using last optimized model
accuracy: 0.911511, std(0.0088413))

```

[Parallel(n\_jobs=6)]: Done 3 out of 3 | elapsed: 0.6s finished

### 0.3.1 Eval Model

```
In [27]: from sklearn.model_selection import train_test_split
        from sklearn.metrics import roc_curve, accuracy_score, precision_recall_curve, auc

        X_train, X_test, Y_train, Y_test = train_test_split(feature_mat, label_mat, test_size=0.3
        # calculate the fpr and tpr for all thresholds of the classification

        fitted = pipeline.fit(X_train, Y_train)
        probs = fitted.predict_proba(X_test)
        preds = probs[:, 1]
        preds_train = fitted.predict_proba(X_train)[:, 1]
        fpr, tpr, threshold = roc_curve(Y_test, preds)
        fpr_train, tpr_train, threshold_train = roc_curve(Y_train, preds_train)
        roc_auc = auc(fpr, tpr)
        roc_auc_train = auc(fpr_train, tpr_train)
        precision, recall, ths = precision_recall_curve(Y_test, preds)
        precision_train, recall_train, ths_train = precision_recall_curve(Y_train, preds_train)

In [28]: from bokeh.models.ranges import Range1d
        #print "auc: %.2g, on train: %.2g" %(roc_auc, roc_auc_train)
        roc_df = pd.DataFrame({"fpr":fpr, "tpr":tpr}).set_index("fpr")
        pr_df = pd.DataFrame({"precision": precision, "recall":recall}).set_index("recall")
        roc_df["diag"] = roc_df.index
        pr_df["random"] = pr_df.precision.iloc[0]

        # roc curve
        roc_f = figure(width=400, height=400, title="roc, auc: %.2g, on train: %.2g" %(roc_auc,
        roc_f.xaxis.axis_label = "tpr"
        auc_range= Range1d(0,1)
        roc_f.x_range = auc_range
        roc_f.y_range = auc_range
        roc_f.yaxis.axis_label = "fpr"
        roc_f.cross(fpr, tpr, size=5)
        roc_f.line(fpr, tpr, legend="roc")
        roc_f.circle(fpr_train, tpr_train, size=5, color="red", line_width=1)
        roc_f.line(fpr_train, tpr_train, color="red", legend="roc on train")
        roc_f.line([0,1], [0,1], color="grey")

        # pr curve
        pr_f = figure(width=400, height=400, title="PR curve")
        pr_f.xaxis.axis_label = "recall"
        pr_f.yaxis.axis_label = "precision"
        pr_f.cross(recall, precision, size=5)
        pr_f.line(recall, precision, legend="PR")
```

```

pr_f.circle(recall_train,precision_train,size=5,color="red", line_width=1)
pr_f.line(recall_train,precision_train,color="red",legend="PR on train")

show(row(
    pr_f,
    roc_f
))

```

### 0.3.2 Feature Importance

```

In [11]: feature_imp = pd.DataFrame({"importance":model.feature_importances_}).set_index(feature
feature_imp.sort_values(by="importance",ascending=False)

```

```

Out[11]:
importance
max_attribute4      0.111504
max_attribute5      0.103783
dft_p0_valattribute7  0.062484
max_attribute2      0.060075
max_attribute8      0.053328
max_attribute7      0.051345
dft_p0_valattribute4  0.048827
mean_attribute9      0.046718
mean_attribute7      0.039789
dft_p0_indattribute4  0.039782
max_attribute6      0.039444
mean_dt2_attribute2  0.037939
max_attribute9      0.035935
mean_attribute6      0.035741
dft_p0_indattribute9  0.032395
mean_dt_attribute2   0.030649
mean_attribute2      0.028052
dft_p0_valattribute6  0.024627
max_dt2_attribute2   0.020886
dft_p0_indattribute6  0.018570
mean_attribute4      0.017630
dft_p0_valattribute9  0.017442
dft_p0_indattribute7  0.012758
max_dt_attribute2    0.011510
mean_attribute8      0.010386
mean_attribute5      0.008401
dft_p0_indattribute5  0.000000
dft_p0_valattribute5  0.000000

```

### 0.3.3 Hyperparameter optimisation

```

In [12]: # model : GradientBoostingClassifier, parameters:
#loss : {deviance, exponential},
#learning_rate : float, optional (default=0.1)

```

```

#n_estimators : int (default=100)
#max_depth : integer, optional (default=3)
#min_samples_split : int, float, optional (default=2)
grids=dict()
XDB_param_grid = {
    #"model__loss": ["deviance", 'exponential'],
    "model__learning_rate" : [1e-3,0.01, 0.1],
    "model__n_estimators" : [10, 50, 100, 150],
    "model__max_depth" : [5,10,15],
    "model__min_samples_split" : [5,10,20]
}
grids[GradientBoostingClassifier] = XDB_param_grid

In [13]: # model : RandomForestClassifier, parameters:
# n_estimators : int (default=100)
# criterion : "gini","entropy"
# max_features : auto , fraction
# max_depth : integer, optional (default=3)
# min_samples_split : int, float, optional (default=2)

RF_param_grid = {
    #"model__criterion": ["gini", "entropy"],
    "model__n_estimators" : [75,100,150,200],
    #"model__max_features" : ["auto",0.5,0.25,0.1],
    "model__max_depth" : [2,5,10,20],
    "model__min_samples_split" : [5,10,20]
}
grids[RandomForestClassifier] = RF_param_grid

In [14]: # C :penalty
# kernel : linear, poly, rbf, sigmoid, precomputed
SVC_param_grid = {
    'model__C': [1e-7,1e-6,1e-5,0.1],
    "model__kernel": ["rbf","linear"],
    #"model__degree" : [1,3,5], # polynomial degrees
    "model__gamma" : ["auto"], # kernel coef (rbf)
    #"coef0" # for poly, signmoid
    "model__tol" : [1e-7,1e-6,1e-5, 1e-4,1e-3]
}
grids[SVC] = SVC_param_grid

In [15]: m = type(dict(pipeline.steps)["model"])
param_grid=grids[m]

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

kfold = StratifiedKFold(n_splits=6, shuffle=True)

```

```
grid_search = GridSearchCV(pipeline, param_grid, scoring="accuracy", n_jobs=-1, verbose=1)
grid_result = grid_search.fit(feature_mat, label_mat)
```

```
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
#for mean, stdev, param in zip(means, stds, params):
#    print("%f (%f) with: %r" % (mean, stdev, param))
```

Fitting 6 folds for each of 48 candidates, totalling 288 fits

```
[Parallel(n_jobs=-1)]: Done 34 tasks      | elapsed:    4.6s
[Parallel(n_jobs=-1)]: Done 184 tasks     | elapsed:   32.8s
[Parallel(n_jobs=-1)]: Done 288 out of 288 | elapsed:   54.1s finished
```

Best: 0.926659 using {'model\_\_min\_samples\_split': 5, 'model\_\_max\_depth': 10, 'model\_\_n\_estimators': 100}

```
In [20]: from tpot import TPOTClassifier
         pipeline_optimizer = TPOTClassifier(
             generations=2,
             population_size=20,
             #generations=10, # the more generation, the more optimized you get
             #population_size=200,
             num_cv_folds=4,
             scoring="accuracy",
             random_state=42,
             verbosity=2)

         pipeline_optimizer.fit(feature_mat, label_mat)
         print pipeline_optimizer.score(feature_mat, label_mat)
         #pipeline_optimizer.export('tpot_longrun_exported_pipeline.py')
```

Optimization Progress: 33%| | 20/60 [00:29<00:30, 1.30pipeline/s] Optimization Progress: 3%|

Generation 1 - Current best internal CV score: 0.955780798458

Optimization Progress: 65%| | 39/60 [01:39<01:12, 3.48s/pipeline] Optimization Progress: 37%|

Generation 2 - Current best internal CV score: 0.956938155554

Optimization Progress: 65%| | 39/60 [01:39<01:12, 3.48s/pipeline] Optimization Progress: 37%|

```
Best pipeline: ExtraTreesClassifier(input_matrix, 17, 0.9899999999999999)
1.0
```

## 0.4 Serialization

```
In [19]: model_name = "device"

        from sklearn.externals import joblib
        import pickle
        joblib.dump(pipeline_optimizer._fitted_pipeline, 'model_%s.pkl' % model_name)
        features = feature_set.columns
        with open('model_%s.feats' % model_name, "w+") as f:
            pickle.dump(features,f)

In [ ]:
```