Bosch Production Line Performance

Pujan Malavia

Company Information

Robert Bosch GmbH, or Bosch, is a German multinational engineering and technology company headquartered in Gerlingen, near Stuttgart, Germany. The company was founded by Robert Bosch in Stuttgart in 1886. Bosch is 92% owned by Robert Bosch Stiftung. Bosch's core operating areas are spread across four business sectors; mobility (hardware and software), consumer goods (including household appliances and power tools), industrial technology (including drive and control) and energy and building technology.

https://en.wikipedia.org/wiki/Robert_Bosch_GmbH (https://en.wikipedia.org/wiki/Robert_Bosch_GmbH)

Challenge

A good chocolate soufflé is decadent, delicious, and delicate. But, it's a challenge to prepare. When you pull a disappointingly deflated dessert out of the oven, you instinctively retrace your steps to identify at what point you went wrong. Bosch, one of the world's leading manufacturing companies, has an imperative to ensure that the recipes for the production of its advanced mechanical components are of the highest quality and safety standards. Part of doing so is closely monitoring its parts as they progress through the manufacturing processes.

Because Bosch records data at every step along its assembly lines, they have the ability to apply advanced analytics to improve these manufacturing processes. However, the intricacies of the data and complexities of the production line pose problems for current methods. In this competition, Bosch is challenging Kagglers to predict internal failures using thousands of measurements and tests made for each component along the assembly line. This would enable Bosch to bring quality products at lower costs to the end user.

https://www.kaggle.com/c/bosch-production-line-performance (https://www.kaggle.com/c/bosch-production-line-performance)

Import Libraries

In [203]:
```
from __future__ import division

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb

from pandas.core.frame import DataFrame

# estimator imports
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import ExtraTreesClassifier, RandomForestClassifier, Gra
dientBoostingClassifier
from sklearn.svm import LinearSVC

# feature manipulation and preprocessing
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_extraction import DictVectorizer

# sampling, grid search, and reporting
from sklearn.cross_validation import StratifiedShuffleSplit
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score

import os

%matplotlib inline
```

Defining classes

In [204]:
```
def nan_evaluation(df, axis=1, method="all"):
    """
        Evaluate all features or rows that have NaN values.

        Parameters
        ----------
        df : {pandas.DataFrame}
            dataframe to evaluate

        axis : {int}
            0 => return rows, 1 => return columns; rows is slow for large data
frames

        method : {string}
            specify whether to return rows/columns with all or some NaN value
s.
    """
    methods = {
        "all": lambda x: np.all(x),
        "some": lambda x: np.any(x)
    }

    if axis == 1:
        return [col for col in df.columns if methods[method](df[col].isnull
())]

    return [row for row in df.index if methods[method](df.iloc[row][1:-1].isnu
ll())]
```

In [205]:
```
def hit_score(y_true, y_pred, hit_class=1):
    """
        Determine the proportion of 'hits' in the predicted set,
        in relation to the number of 'hits' in the true set.

        Parameters
        ----------
        y_true : {numpy.1darray}
            true data points; usually corresponds to test set

        y_pred : {numpy.1darray}
            estimator outcome

        hit_class : {int}
            class value used to calculate number of hits
    """
    hit_idx = [idx for idx, val in enumerate(y_true) if val == hit_class]
    hits = 0
    for pred_idx, pred_val in enumerate(y_pred):
        if pred_idx in hit_idx and pred_val == hit_class:
            hits += 1

    return (hits / len(hit_idx))*100
```

In [206]:
```
def scoring_report(y_true, y_pred, estimator_type=""):
    """
        Helper function to output scores for binary classification prediction.

        Parameters
        ----------
        y_true : {numpy.1darray}
            true data points; usually corresponds to test set

        y_pred : {numpy.1darray}
            estimator outcome

        estimator_type : {string}
            type of estimator (used for string formatting)
    """
    print ("%s Accuracy Score: %s" % (estimator_type, accuracy_score(y_true=y_
true, y_pred=y_pred) * 100))
    print ("%s Hit Score: %s" % (estimator_type, hit_score(y_true=y_true, y_pr
ed=y_pred, hit_class=1)))
    print ("%s Classification Report:" % estimator_type)
    print (classification_report(y_true=y_true, y_pred=y_pred, digits=3))
```

Import/Explore Dataset

In [207]:
```
df_numeric = pd.read_csv("C:/Users/puj83/OneDrive/Portfolio/Bosch_Production_L
ine/train_numeric.csv")
```

In [208]:
```
len(df_numeric)
```

Out[208]: 1183747

In [209]:
```
len(df_numeric.columns)
```

Out[209]: 970

In [210]:
```
df_numeric.head()
```

Out[210]:

	Id	L0_S0_F0	L0_S0_F2	L0_S0_F4	L0_S0_F6	L0_S0_F8	L0_S0_F10	L0_S0_F12	L0
0	4	0.030	-0.034	-0.197	-0.179	0.118	0.116	-0.015	-0.
1	6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	7	0.088	0.086	0.003	-0.052	0.161	0.025	-0.015	-0.
3	9	-0.036	-0.064	0.294	0.330	0.074	0.161	0.022	0.1
4	11	-0.055	-0.086	0.294	0.330	0.118	0.025	0.030	0.1

5 rows × 970 columns

In [211]:
```
# all columns that have strictly nan values
len(nan_evaluation(df_numeric, axis=1))
```

Out[211]: 0

In [212]:
```
# all columns that have some nan values
len(nan_evaluation(df_numeric, axis=1, method="some"))
```

Out[212]: 968

In [216]:
```
df_numeric_mean = df_numeric.fillna(df_numeric.mean(), inplace=True)
```

In [219]:
```
features = list(set(df_numeric_mean.columns) - set(["Id", "Response"]))
```

In [220]:
```
X = df_numeric_mean[features].values
y = df_numeric_mean["Response"].values
```

Splitting into train/test set

In [221]:
```
X_train = X[:int(0.8*len(X))]
X_test = X[int(0.8*len(X)):]
y_train = y[:int(0.8*len(y))]
y_test = y[int(0.8*len(y)):]
```

In [222]:
```
# proportion of positives (1) to negatives (0) in train and test sets
print ("positive proportion in train: {}%".format((len(y_train[y_train==1]) /
len(y_train))*100))
print ("negative proportion in train: {}%".format((len(y_train[y_train==0]) /
len(y_train))*100))
print ("positive proportion in test: {}%".format((len(y_test[y_test==1]) / len
(y_test))*100))
print ("negative proportion in test: {}%".format((len(y_test[y_test==0]) / len
(y_test))*100))
```

```
positive proportion in train: 0.578776912704053%
negative proportion in train: 99.42122308729596%
positive proportion in test: 0.5904963041182683%
negative proportion in test: 99.40950369588172%
```

Feature Selection

Extra Trees Classifier

In [223]:
```
# only use 10 estimators (default) for when we want to run feature selection
xt_clf = ExtraTreesClassifier(n_estimators=10, verbose=2)
```

In [224]:
```
xt_clf.fit(X_train, y_train)
```

```
building tree 1 of 10

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   17.4s remaining:    0.
0s

building tree 2 of 10
building tree 3 of 10
building tree 4 of 10
building tree 5 of 10
building tree 6 of 10
building tree 7 of 10
building tree 8 of 10
building tree 9 of 10
building tree 10 of 10

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:  3.3min finished
```

Out[224]:
```
ExtraTreesClassifier(bootstrap=False, class_weight=None, criterion='gini',
           max_depth=None, max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=2, warm_start=False)
```

In [225]:
```
prediction = xt_clf.predict(X_test)
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.
0s
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    3.0s finished
```

In [226]:
```
print ("Extra Trees Classifier Accuracy Score: %s" % (accuracy_score(y_true=y_
test, y_pred=prediction) * 100))
print ("Extra Trees Hit Score: %s" % hit_score(y_true=y_test, y_pred=predictio
n, hit_class=1))
print ("Extra Trees Classification Report:")
print (classification_report(y_true=y_test, y_pred=prediction, digits=3))
```

```
Extra Trees Classifier Accuracy Score: 99.4116156283
Extra Trees Hit Score: 0.7868383404864091
Extra Trees Classification Report:
             precision    recall  f1-score   support

          0      0.994     1.000     0.997    235352
          1      0.647     0.008     0.016      1398

avg / total      0.992     0.994     0.991    236750
```

In [227]:
```
# use pre-written feature selection methods to determine
xt_sfm = SelectFromModel(xt_clf, prefit=True)
```

In [228]:
```
xt_sfm_support = xt_sfm.get_support()
```

In [229]:
```
xt_sfm_features = list(map(lambda y: y[1], filter(lambda x: xt_sfm_support[x[0
]], enumerate(features))))
```

Random Forest

In [230]:
```
rfc_clf = RandomForestClassifier(n_estimators=10, verbose=2)
```

In [231]:
```
rfc_clf.fit(X_train, y_train)
```

```
building tree 1 of 10

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   37.0s remaining:    0.
0s

building tree 2 of 10
building tree 3 of 10
building tree 4 of 10
building tree 5 of 10
building tree 6 of 10
building tree 7 of 10
building tree 8 of 10
building tree 9 of 10
building tree 10 of 10

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:  6.3min finished
```

Out[231]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=2,
            warm_start=False)
```

In [232]:
```
prediction = rfc_clf.predict(X_test)
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.
0s
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    2.2s finished
```

In [233]:
```
print ("Random Forest Classifier Accuracy Score: %s" % (accuracy_score(y_true=
y_test, y_pred=prediction) * 100))
print ("Random Forest Hit Score: %s" % hit_score(y_true=y_test, y_pred=predict
ion, hit_class=1))
print ("Random Forest Classification Report:")
print (classification_report(y_true=y_test, y_pred=prediction, digits=3))
```

```
Random Forest Classifier Accuracy Score: 99.4175290391
Random Forest Hit Score: 2.432045779685265
Random Forest Classification Report:
             precision    recall  f1-score   support

          0      0.994     1.000     0.997    235352
          1      0.694     0.024     0.047      1398

avg / total      0.992     0.994     0.991    236750
```

In [234]:
```
rf_sfm = SelectFromModel(rfc_clf, prefit=True)
```

In [235]:
```
rf_sfm_support = rf_sfm.get_support()
```

In [236]:
```
rf_sfm_features = list(map(lambda y: y[1], filter(lambda x: rf_sfm_support[x[0
]], enumerate(features))))
```

Gradient Boosting Machine (GBM)

In [237]:
```
gb_clf = GradientBoostingClassifier(n_estimators=10, verbose=2)
```

In [238]:
```
gb_clf.fit(X_train, y_train)
```

```
      Iter       Train Loss   Remaining Time
         1           0.0695            4.76m
         2           0.0669            4.19m
         3           0.0662            3.58m
         4           0.0658            2.98m
         5           0.0656            2.53m
         6           0.0654            2.02m
         7           0.0652            1.51m
         8           0.0651           59.95s
         9           0.0649           29.97s
        10           0.0648            0.00s
```

Out[238]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=10,
              presort='auto', random_state=None, subsample=1.0, verbose=2,
              warm_start=False)
```

In [239]:
```
prediction = gb_clf.predict(X_test)
```

In [240]:
```
print ("Gradient Boost Accuracy Score: %s" % (accuracy_score(y_true=y_test, y_
pred=prediction) * 100))
print ("Gradient Boost Hit Score: %s" % hit_score(y_true=y_test, y_pred=predic
tion, hit_class=1))
print ("Gradient Boost Classification Report:")
print (classification_report(y_true=y_test, y_pred=prediction, digits=3))
```

```
Gradient Boost Accuracy Score: 99.4225976769
Gradient Boost Hit Score: 4.291845493562231
Gradient Boost Classification Report:
             precision    recall  f1-score   support

          0      0.994     1.000     0.997    235352
          1      0.674     0.043     0.081      1398

avg / total      0.992     0.994     0.992    236750
```

Model Training

Gradient Boosting Machine (GBM)

In [241]:
```
# gb_clf = GradientBoostingClassifier(n_estimators=50, verbose=2)
```

In [244]:
```
# gb_clf.fit(X_new_train, y_train)
```

In [245]:
```
# prediction = gb_clf.predict(X_new_test)
```

In [246]:
```
# scoring_report(y_test, prediction, estimator_type="Gradient Boost")
```

Support Vector Machine (SVM)

In [247]:
```
# lsvm = LinearSVC(verbose=2)
```

In [248]:
```
# lsvm.fit(X_new_train, y_train)
```

In [249]:
```
# prediction = lsvm.predict(X_new_test)
```

In [250]:
```
# scoring_report(y_test, prediction, estimator_type="LinearSVC")
```

Random Forest (RF)

In [257]:
```
rf_clf = RandomForestClassifier(n_estimators=10, verbose=2, oob_score=True)
```

In [260]:
```
X_new = df_numeric_mean[features_iter_1].values
```

In [261]:
```
X_new_train = X_new[:int(0.8*len(X_new))]
X_new_test = X_new[int(0.8*len(X_new)):]
```

In [258]:
```
rf_clf.fit(X_new_train, y_train)
```

```
building tree 1 of 10
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   31.6s remaining:    0.
0s
building tree 2 of 10
building tree 3 of 10
building tree 4 of 10
building tree 5 of 10
building tree 6 of 10
building tree 7 of 10
building tree 8 of 10
building tree 9 of 10
building tree 10 of 10
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:  5.2min finished
C:\Users\puj83\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:453: Us
erWarning: Some inputs do not have OOB scores. This probably means too few tr
ees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
C:\Users\puj83\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:458: Ru
ntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

Out[258]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=True, random_state=None, verbose=2, warm_start=False)
```

In [266]:
```
prediction = rf_clf.predict(X_new_test)
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.
0s
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    1.5s finished
```

In [267]:
```
scoring_report(y_test, prediction, estimator_type="Random Forest")
```

```
Random Forest Accuracy Score: 99.4128827878
Random Forest Hit Score: 2.7181688125894135
Random Forest Classification Report:
             precision    recall  f1-score   support

          0      0.994     1.000     0.997    235352
          1      0.559     0.027     0.052      1398

avg / total      0.992     0.994     0.991    236750
```