Evaluation Part 2

Uninteresting stuff we already talked about

In [4]:
```
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import OrdinalEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import tree
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE

maintenance = pd.read_csv("predictive_maintenance_prepared.csv")

X = maintenance[maintenance.columns[0:-1]]
y = maintenance['Target']

train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=20, test_size=0.3, strati

rf_model = RandomForestClassifier(random_state=20)
```

Predicting via RandomForest

In [17]:
```
rf_model.fit(train_X, train_y)
pred_y = rf_model.predict(test_X)
accuracy_score = metrics.accuracy_score(pred_y, test_y)

print('Accuracy: {:.2%}'.format(metrics.accuracy_score(test_y, pred_y)))
print('Recall: {:.2%}'.format(metrics.recall_score(test_y, pred_y)))
print('Precision: {:.2%}'.format(metrics.precision_score(test_y, pred_y)))
print('F1 Score: {:.2%}'.format(metrics.f1_score(test_y, pred_y)))

print('-----')
print('Ratio')
train_y.value_counts()
```

```
Accuracy: 98.23%
Recall: 52.94%
Precision: 91.53%
F1 Score: 67.08%
------
Ratio
```

Out[17]:
```
0    6763
1     237
Name: Target, dtype: int64
```

We can see that even though the Accuracy is quite high, the Recall and F1 score are considerably low. As we work with machines a false negative would be detrimental, so we have to try to get Recall as high as possible.

Problem is our "working to not working machines" ratio is really low ~(96.496% / 3,504%). We have to do a little bit of *magic*.

Random Oversampling

As we could see in the example before, the amount of faulty machines is way smaller than the amount of working machines. To fix this, first we are going to try to oversample.

In [14]:
```
oversampler = RandomOverSampler(sampling_strategy=1,random_state=20)
over_X, over_y = oversampler.fit_resample(train_X, train_y)

rf_model = RandomForestClassifier(random_state=20)
rf_model.fit(over_X, over_y)
pred_y = rf_model.predict(test_X)

print('Accuracy: {:.2%}'.format(metrics.accuracy_score(test_y, pred_y)))
print('Precision: {:.2%}'.format(metrics.precision_score(test_y, pred_y)))
print('Recall: {:.2%}'.format(metrics.recall_score(test_y, pred_y)))
print('F1: {:.2%}'.format(metrics.f1_score(test_y, pred_y)))

over_y.value_counts()
```

```
Accuracy: 98.33%
Precision: 86.11%
Recall: 60.78%
F1: 71.26%
```

Out[14]:
```
0    6763
1    6763
Name: Target, dtype: int64
```

The results are way better. Does it really matter? No. Just oversampling or undersampling would of course not be enough so what is the solution?

Over- and undersampling

Sure, let's see what happens.

In [15]:
```
oversampler = RandomOverSampler(sampling_strategy=0.4,random_state=20)
ounder_X, ounder_y = oversampler.fit_resample(train_X, train_y)

undersampler = RandomUnderSampler(sampling_strategy=1, random_state=22)
ounder_X, ounder_y = undersampler.fit_resample(ounder_X, ounder_y)

ounder_y.value_counts()
```

Out[15]:
```
0    2705
1    2705
Name: Target, dtype: int64
```

In [18]:
```
rf_model = RandomForestClassifier(random_state=20)
rf_model.fit(ounder_X, ounder_y)
y_pred = rf_model.predict(test_X)

print('Accuracy: {:.2%}'.format(metrics.accuracy_score(test_y, y_pred)))
print('Precision: {:.2%}'.format(metrics.precision_score(test_y, y_pred)))
print('Recall: {:.2%}'.format(metrics.recall_score(test_y, y_pred)))
print('F1: {:.2%}'.format(metrics.f1_score(test_y, y_pred)))
```

```
Accuracy: 97.80%
Precision: 66.07%
Recall: 72.55%
F1: 69.16%
```

An increase of ~20% to the Recall is very nice for the "cost" of just under a 1% in the Accuracy.