Boosting for Bearing Fault Classification

Biswajit Sahoo

Boosting is slow learning procedure. In boosting, a tree is not allowed to grow fully. Rather, the depth is fixed to a small number. In some cases, a tree is build with only one branch and two nodes. But subsequent trees are built on top of it. Further, contributions from subsequent trees are also shrinked and added to previously built tree. Though an individual tree might preform badly for the classification, combination of many trees perform surprisingly well. For more details regarding the algorithm, readers can refer to this excellent book.

In this post we will apply boosting to classify multiclass bearing fault.

Description of data

Detailed discussion of how to prepare the data and its source can be found in this post. Here we will only mention about different classes of the data. There are 12 classes and data for each class are taken at a load of 1hp. The classes are:

- C1 : Ball defect (0.007 inch)
- C2 : Ball defect (0.014 inch)
- C3 : Ball defect (0.021 inch)
- C4 : Ball defect (0.028 inch)
- C5 : Inner race fault (0.007 inch)
- C6 : Inner race fault (0.014 inch)
- C7 : Inner race fault (0.021 inch)
- C8 : Inner race fault (0.028 inch)
- C9 : Normal
- C10 : Outer race fault (0.007 inch, data collected from 6 O'clock position)
- C11 : Outer race fault (0.014 inch, 6 O'clock)
- C12 : Outer race fault (0.021 inch, 6 O'clock)

Important Note: In the CWRU website, sampling frequency for the normal data is not mentioned. Most research paper take it as 48k. Some authors also consider it as being taken at a sampling frequency of 12k. Some other authors just use it without ever mentioning its sampling frequency. In our application we only need segment of normal data of length 1024. So we will use the normal data segments available at the website without going into the discussion of sampling frequency. Still, to be on the safer side, we will show results including the normal data as a class as well as excluding it.

When we exclude normal data, we won't consider "C9" class and study the rest 11 fault classes. At that time "C09", "C10", and "C11" will correspond to outer race faults of fault depth 0.007, 0.014, and 0.021 inch respectively.

Codes

```
library(reticulate)
use_condaenv("r-reticulate")
```

First download the data from here. Save the data in a folder and read it from that folder.

```
data_wav_energy = read.csv('feature_wav_energy8_12k_1024_load_1.csv',
                           header = T)
# Change the above line to include your folder that contains data
```

```r
set.seed(1)
index = c(sample(1:115,35),sample(116:230,35), sample(231:345,35),
          sample(346:460,35),sample(461:575,35),sample(576:690,35),
          sample(691:805,35),sample(806:920,35),sample(921:1035,35),
          sample(1036:1150,35),sample(1151:1265,35),sample(1266:1380,35))

train_data = data_wav_energy[-index,]
test_data = data_wav_energy[index,]

# Shuffle data
train_data = train_data[sample(nrow(train_data)),]
test_data = test_data[sample(nrow(test_data)),]
```

It should be noted that for some of the deterministic techniques, shuffling of data is not required. But some other techniques like deep learning require the data to be shuffled for better training. So as a recipe we always shuffle data whether the method is deterministic or not. This doesn't hurt either for a deterministic technique.

We will perform gradient boosting using 'gbm' package.

```r
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```r
boosting_fit = gbm(fault~., train_data, distribution = "multinomial",
                   interaction.depth = 3, n.trees = 500)
pred_boosting = predict(boosting_fit, test_data,n.trees = 500,
                        type = "response")
dim(pred_boosting)
```

```
## [1] 420  12   1
```

Note that boosting prediction is in terms of a matrix. The rows correspond to data points and columns correspond to fault classes. Each row of this matrix gives the probability of the observation being in the class corresponding to the column. The data point is classified to a category for which it has highest probability of occurrence. This can be done by the following code

```r
prediction = colnames(pred_boosting)[apply(pred_boosting, 1, which.max)]
# Confusion matrix
test_confu = table(test_data$fault, prediction)
```

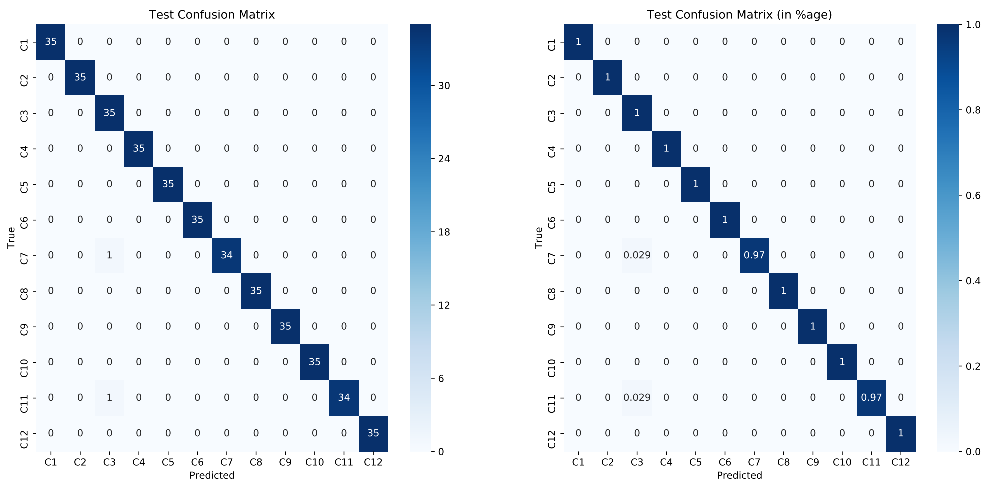
```python
import seaborn as sns
import matplotlib.pyplot as plt
fault_type = ['C1','C2','C3','C4','C5','C6','C7','C8','C9','C10','C11','C12']
plt.figure(1,figsize=(18,8))
plt.subplot(121)
sns.heatmap(r.test_confu, annot = True,
xticklabels=fault_type, yticklabels=fault_type, cmap = "Blues")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x0000000025F8FDD8>
```

```python
plt.title('Test Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.subplot(122)
sns.heatmap(r.test_confu/35, annot = True,
xticklabels=fault_type, yticklabels=fault_type, cmap = "Blues")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000000270400F0>
```

```
plt.title('Test Confusion Matrix (in %age)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```




```
overall_test_accuracy = sum(diag(test_confu))/420
sprintf("Overall Test Accuracy: %.4f", overall_test_accuracy*100)
```

```
## [1] "Overall Test Accuracy: 99.5238"
```

We will also show the results excluding the normal data. The results are as below.

```
data_without_normal = read.csv("feature_wav_energy8_12k_1024_load_1.csv",
                               header = T, nrows = 1265)
# Change the above line to include your folder that contains data
set.seed(1)
index = c(sample(1:115,35),sample(116:230,35), sample(231:345,35),
          sample(346:460,35),sample(461:575,35),sample(576:690,35),
          sample(691:805,35),sample(806:920,35),sample(921:1035,35),
          sample(1036:1150,35),sample(1151:1265,35))

train_new = data_without_normal[-index,]
test_new = data_without_normal[index,]

# Shuffle data
train_data_new = train_new[sample(nrow(train_new)),]
test_data_new = test_new[sample(nrow(test_new)),]
```

```
boosting_fit_new = gbm(fault~., train_data_new, distribution = "multinomial",                        in
pred_boosting_new = predict(boosting_fit_new, test_data_new,n.trees = 500,
                            type = "response")
prediction_new = colnames(pred_boosting_new)[apply(pred_boosting_new, 1, which.max)]
# Confusion matrix
test_confu_new = table(test_data_new$fault, prediction_new)
```

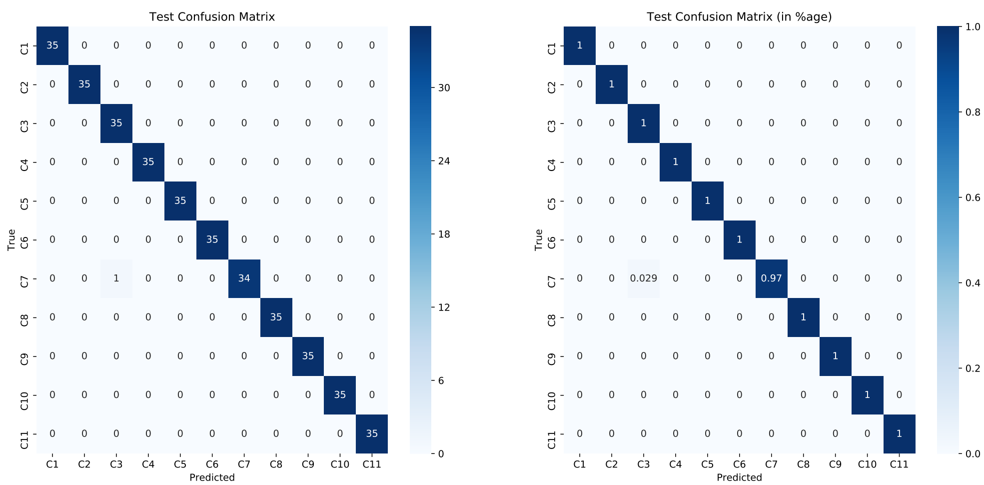
```python
import seaborn as sns
import matplotlib.pyplot as plt
fault_type = ['C1','C2','C3','C4','C5','C6','C7','C8','C9','C10','C11']
plt.figure(1,figsize=(18,8))
plt.subplot(121)
sns.heatmap(r.test_confu_new, annot = True,
xticklabels=fault_type, yticklabels=fault_type, cmap = "Blues")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x000000002875AD30>
```

```python
plt.title('Test Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.subplot(122)
sns.heatmap(r.test_confu_new/35, annot = True,
xticklabels=fault_type, yticklabels=fault_type, cmap = "Blues")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000000288D86A0>
```

```python
plt.title('Test Confusion Matrix (in %age)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```




```r
overall_test_accuracy_new = sum(diag(test_confu_new))/385
sprintf("New overall Test Accuracy: %.4f", overall_test_accuracy_new*100)
```

```
## [1] "New overall Test Accuracy: 99.7403"
```

There are several hyper-parameters in this model, number of trees, interaction depth, and shrinkage. Optimal values for each of these parameters can be obtained by cross validation. In this post, we have chosen some of the commonly used values. Other values can also be tried.

To see results of other techniques applied to public condition monitoring datasets, visit this page.

Last updated: 8^{th} July, 2019