Random Forest for Bearing Fault Classification

Biswajit Sahoo

As we have mentioned in a previous post, random forest is very similar to bagging. As in bagging, many different trees are also grown on bootstrap samples in random forest. The difference between the two methods is the number of variable considered while doing a split in the tree. In bagging, to make a split, all predictor variables are considered and the best one is selected with a suitable value. But in random forest only a random sample of few variables are chosen for each split. If training data have p features, usually $\sqrt{p}$ feature are chosen at random at each split and out of those best one is selected. This helps in reducing correlation between trees. Due to this, usually random forest produces better results than bagging.

In this post, we will use random forest to classify different bearing faults. The dataset has 8 variables corresponding to wavelet energy value in each packet. In random forest we will use 4 random features at the time of split.

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 10 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 : Inner race fault (0.007 inch)
- C5 : Inner race fault (0.014 inch)
- C6 : Inner race fault (0.021 inch)
- C7 : Normal
- C8 : Outer race fault (0.007 inch, data collected from 6 O'clock position)
- C9 : Outer race fault (0.014 inch, 6 O'clock)
- C10 : Outer race fault (0.021 inch, 6 O'clock)

Codes

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

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

```r
data_wav_energy = read.csv('feature_wav_energy8_48k_2048_load_1.csv',
                           header = T)
# Change the above line to include your folder that contains data
set.seed(1)
index = c(sample(1:230,75),sample(231:460,75), sample(461:690,75),
          sample(691:920,75),sample(921:1150,75),sample(1151:1380,75),
          sample(1381:1610,75),sample(1611:1840,75),sample(1841:2070,75),
          sample(2071:2300,75))

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.

Results of random forest might vary for different iterations. As we have set the seed previously, we won't set another seed.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
forest_fit = randomForest(fault~., train_data, mtry = 4)
pred_forest = predict(forest_fit, test_data)
# Confusion matrix
test_confu = table(test_data$fault, pred_forest)
```

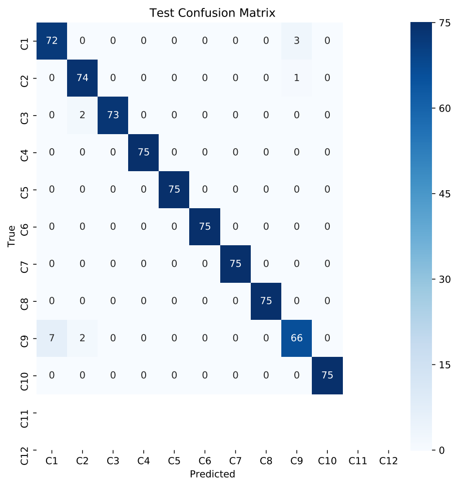
```
import seaborn as sns
import matplotlib.pyplot as plt
fault_type = ['C1','C2','C3','C4','C5','C6','C7','C8','C9','C10','C11','C12']
plt.figure(1,figsize=(18,8))
plt.subplot(121)
sns.heatmap(r.test_confu, annot = True,
xticklabels=fault_type, yticklabels=fault_type, cmap = "Blues")
```

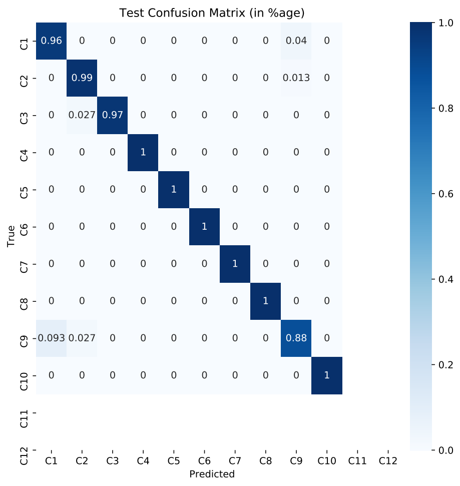
```
## <matplotlib.axes._subplots.AxesSubplot object at 0x000000002239FD30>
```

```
plt.title('Test Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.subplot(122)
sns.heatmap(r.test_confu/75, annot = True,
xticklabels=fault_type, yticklabels=fault_type, cmap = "Blues")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x0000000024B02FD0>
```

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





```
overall_test_accuracy = sum(diag(test_confu))/750
sprintf("Overall Test Accuracy: %.4f", overall_test_accuracy*100)
```

```
## [1] "Overall Test Accuracy: 98.0000"
```

In the above code, we have fixed the number of variables chosen at the time of split to be 4. But a different number of variables other than 4 can also be selection. In the code below we will fit random forest models by choosing variables starting from 1 to 8. Classification accuracy of each of those models is also calculated.

```
accuracy = double(8)
for (i in 1:8){
  temp_mod = randomForest(fault~., train_data, mtry = i)
  temp_pred = predict(temp_mod, test_data)
  accuracy[i] = sum(diag(table(temp_pred, test_data$fault)))/750
}
accuracy
```

```
## [1] 0.9706667 0.9760000 0.9786667 0.9773333 0.9773333 0.9746667 0.9746667
## [8] 0.9680000
```

Note that accuracy with 4 variables is slightly less that what we have calculated before. This variation is due to inherent randomness while fitting random forest models. In practice, it is advantageous to try out several values and pick the best one.

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

Last updated: 8th July, 2019