# Quadratic Discriminant Analysis for Bearing Fault Classification

*Biswajit Sahoo*

In a previous post we had discussed LDA. In LDA, a common covariance matrix is assumed for all classes. But in Quadratic Discriminant Analysis (QDA) different covariance matrices are used for different classes. Thus number of parameters in QDA is much larger than LDA and it is prone to high variance. But if data satisfies the underlying assumptions of QDA and the dataset is not to small, QDA gives better results.

We will postpone the details of the method to a later time. For the time being we direct the interested reader to this excellent book.

We will provide codes in R to show that LDA also works well for bearing fault classification. We will use package 'MASS' to implement QDA.

## 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
library(MASS)
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.

```
qda_fit = qda(fault~., train_data)
pred_train = predict(qda_fit, newdata = train_data)
pred_test = predict(qda_fit, newdata = test_data)
# Confusion matrix
train_confu = table(train_data$fault, pred_train$class)
test_confu = table(test_data$fault, pred_test$class)
```

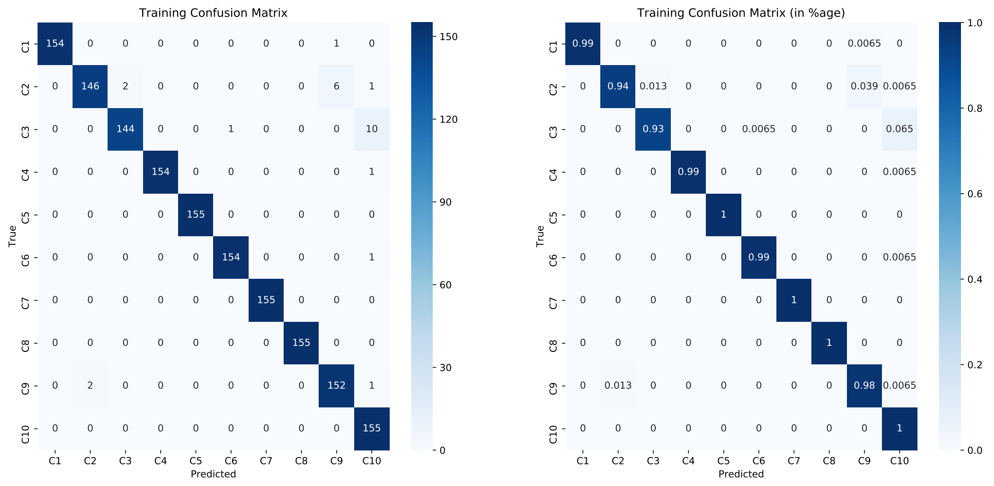
```
import seaborn as sns
import matplotlib.pyplot as plt
fault_type = ['C1','C2','C3','C4','C5','C6','C7','C8','C9','C10']
plt.figure(1,figsize=(18,8))
plt.subplot(121)
sns.heatmap(r.train_confu, annot= True,fmt = "d",
xticklabels=fault_type, yticklabels=fault_type, cmap = "Blues")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000000244D9470>
```

```
plt.title('Training Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.subplot(122)
sns.heatmap(r.train_confu/155, annot= True,
xticklabels=fault_type, yticklabels=fault_type, cmap = "Blues")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x000000002658D4E0>
```

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



| | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 |
|---|---|---|---|---|---|---|---|---|---|---|
| C1 | 154 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| C2 | 0 | 146 | 2 | 0 | 0 | 0 | 0 | 0 | 6 | 1 |
| C3 | 0 | 0 | 144 | 0 | 0 | 1 | 0 | 0 | 0 | 10 |
| C4 | 0 | 0 | 0 | 154 | 0 | 0 | 0 | 0 | 0 | 1 |
| C5 | 0 | 0 | 0 | 0 | 155 | 0 | 0 | 0 | 0 | 0 |
| C6 | 0 | 0 | 0 | 0 | 0 | 154 | 0 | 0 | 0 | 1 |
| C7 | 0 | 0 | 0 | 0 | 0 | 0 | 155 | 0 | 0 | 0 |
| C8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 155 | 0 | 0 |
| C9 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 152 | 1 |
| C10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 155 |

| | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 |
|---|---|---|---|---|---|---|---|---|---|---|
| C1 | 0.99 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.0065 | 0 |
| C2 | 0 | 0.94 | 0.013 | 0 | 0 | 0 | 0 | 0 | 0.039 | 0.0065 |
| C3 | 0 | 0 | 0.93 | 0 | 0 | 0.0065 | 0 | 0 | 0 | 0.065 |
| C4 | 0 | 0 | 0 | 0.99 | 0 | 0 | 0 | 0 | 0 | 0.0065 |
| C5 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| C6 | 0 | 0 | 0 | 0 | 0 | 0.99 | 0 | 0 | 0 | 0.0065 |
| C7 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| C8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| C9 | 0 | 0.013 | 0 | 0 | 0 | 0 | 0 | 0 | 0.98 | 0.0065 |
| C10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

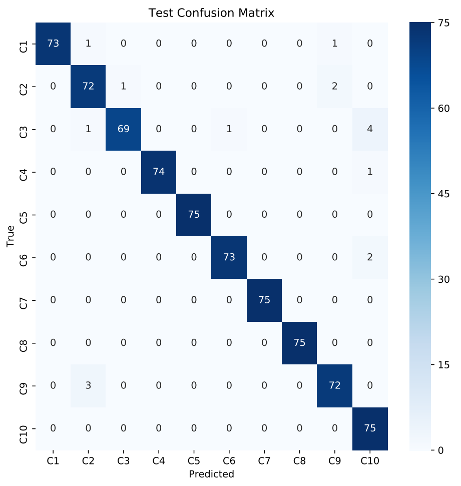
```
plt.figure(2,figsize=(18,8))
plt.subplot(121)
sns.heatmap(r.test_confu, annot = True,
xticklabels=fault_type, yticklabels=fault_type, cmap = "Blues")
```

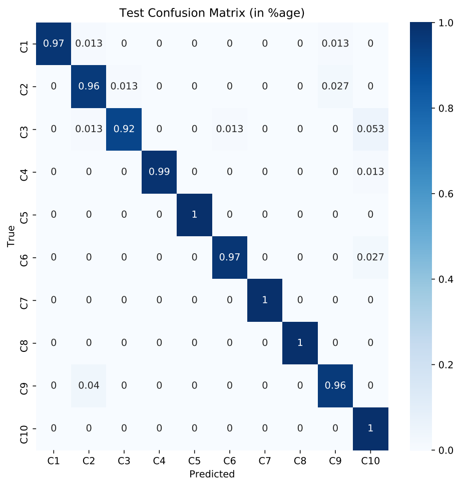
```
## <matplotlib.axes._subplots.AxesSubplot object at 0x0000000026C349E8>
```

```
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 0x000000001F4BB518>
```

```
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: 97.7333"
```

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

Last updated: 7th July, 2019