Multiclass Logistic Regression for Bearing Fault Classification

Biswajit Sahoo

Before discussing multiclass logistic regression, we will briefly mention logistic regression. Logistic regression is a binary classification technique that uses logistic function ($\frac{1}{1+e^{-x}}$) to fit data. Contrary to linear regression where the output is a numerical value, in logistic regression the output is a probability scores of an input belonging to a particular class.

If there are p predictors (also known as independent variables) $X_1, X_2, \ldots, X_p$, in linear regression, the target (dependent variable) y takes the form

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots \beta_p X_p$$

In logistic regression, the target is either 0 (for one class) or 1 (for other class) and $p(x)$ denotes the probability that a point belongs to a particular class. If the probability score exceeds a threshold, it is assigned to one class or other. In logistic regression log odds is modeled as a linear combination of predictors

$$log(\frac{p(x)}{1-p(x)}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots \beta_p X_p$$

After simplification, the above equation will take the form

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots \beta_p X_p)}}$$

Now the task is to learn the parameters $\beta_0, \beta_1, \ldots \beta_p$. This is done by minimizing the negative log-likelihood with respect to training data. Minimizing negative log-likelihood is equivalent to maximizing likelihood.

One point to note here is that, unlike linear regression where a closed form solution exists, logistic regression solution may not converge sometimes. This happens if the classes are well separated from each other. We will not discuss this point further here, rather we will refer the readers to this excellent short article.

As mentioned previously, logistic regression can handle only two classes. For multiclass classification problems there are extensions of logistic regression. One simple strategy is to do one-vs-all classification. This converts multiclass classification problem involving k classes into k binary classification problem. In each of those k binary classification problems, one class is compared against rest all of the classes. For prediction, a point is assigned to a class for which its probability score is maximum.

We will not implement these on our own. Rather we will use R. The beauty of R is that most of the statistical techniques are already implemented in it so that we can just use those for our analysis. In our case we will use 'nnet' package to implement multiclass logistic regression algorithm involving 10 classes.

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

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

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

```
library(nnet)
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.

Now we will apply 'multinom' function that does multiclass logistic regression. We will also use the generic predict function that is used for prediction once the model is fit. Finally we will plot the results.

```
multi_logit = multinom(fault~., data = train_data)
```

```
## # weights:  100 (81 variable)
## initial  value 3569.006894
## iter  10 value 2861.289095
## iter  20 value 2577.713923
## iter  30 value 1890.725181
## iter  40 value 1019.673643
## iter  50 value 513.044278
## iter  60 value 427.984979
## iter  70 value 361.809633
## iter  80 value 322.418441
## iter  90 value 296.304259
## iter 100 value 276.480693
## final  value 276.480693
## stopped after 100 iterations
```

```r
pred_train = predict(multi_logit, newdata = train_data)
pred_test = predict(multi_logit, newdata = test_data)
# Confusion matrix
train_confu = table(train_data$fault, pred_train)
test_confu = table(test_data$fault, pred_test)
```

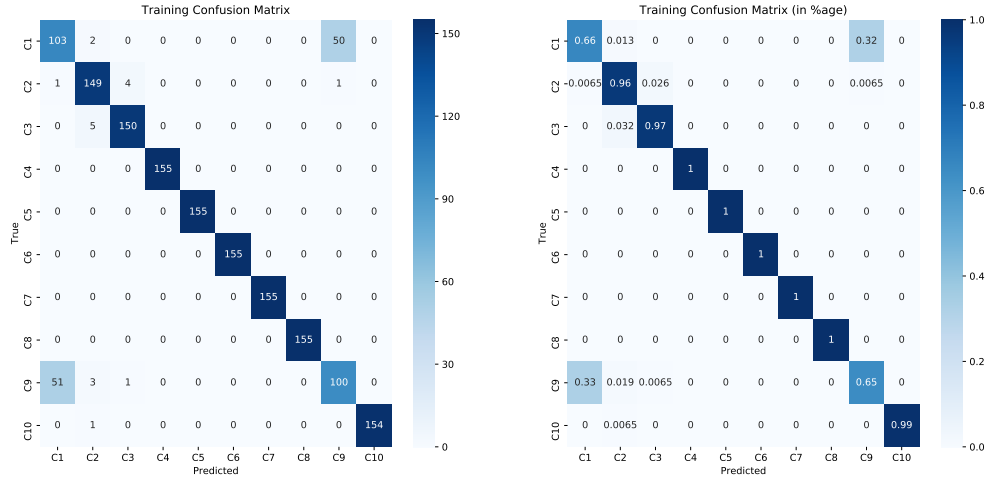
```python
import seaborn as sns
import matplotlib.pyplot as plt
fault_type = ['C1','C2','C3','C4','C5','C6','C7','C8','C9','C10']
plt.figure(1,figsize=(18,8))
plt.subplot(121)
sns.heatmap(r.train_confu, annot= True,fmt = "d",
xticklabels=fault_type, yticklabels=fault_type, cmap = "Blues")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x0000000023119470>
```

```python
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 0x0000000024173278>
```

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



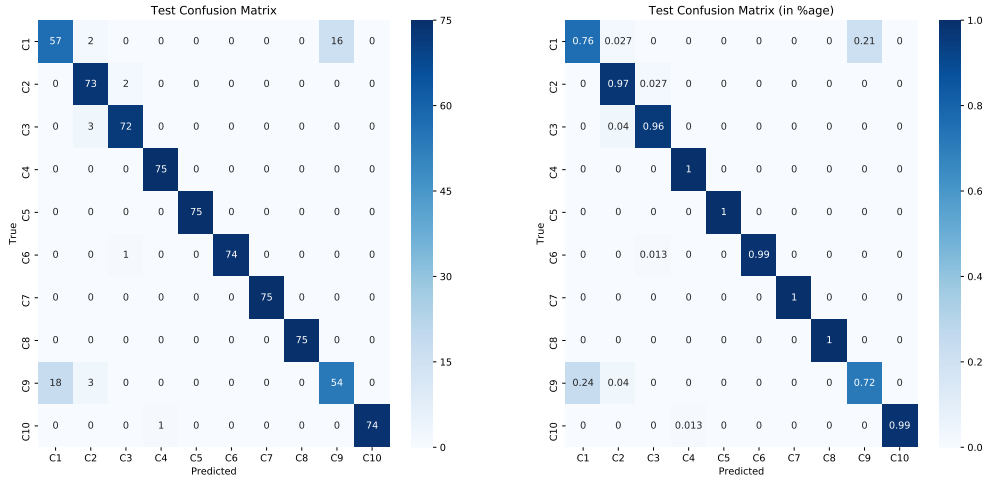

```r
plt.figure(2,figsize=(18,8))
plt.subplot(121)
sns.heatmap(r.test_confu, annot = True,
xticklabels=fault_type, yticklabels=fault_type, cmap = "Blues")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x0000000025874978>
```

```
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 0x00000000241D5908>
```

```
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: 93.8667"
```

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

Last updated: 7^{th} July, 2019