Multiclass classification using SVM on time domain features

Biswajit Sahoo

In this post we will use Case Western Reserve University Bearnig data set for our multiclass classification problem.

Description of data set

A bearing has four major parts: inner race, outer race, rolling element and cage. Fault can occur in any of these components. The CWRU data set contains bearing data consisting of inner race fault, outer race fault and ball defect. A baseline (normal) bearing data with no faults is also available. Some data are collected with a sampling frequency of 12 kHz and some other are collected with 48 kHz. In this study we will only consider data acquired with 48 kHz sampling frequency. The faults have varying fault depths (0.007 inch, 0.014 inch, 0.021 inch). There is also load variation in motor (No load, 1 hp, 2 hp, 3hp). For this study we consider all the case with 1 hp external load.

There are 10 class for this external load (1 hp). 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)

Solution Approach

Our task is to classify these 10 types of fault given time data. There are many approaches to solve this. We will take one known as 'Shallow Approach'. In the age of deep learning these methods are shallow for several reasons. These methods require hand crafted features to be designed and fed into the learning algorithm. Another name for shallow approach is feature based approach. We will use support vector machine (SVM) to do the classification. We will apply other techniques including deep learning techniques in later posts.

We have used time domain features as input to SVM. First data for each fault type are collected and segmented into smaller parts. In our case, one segment for each fault type contains 2048 data points. Then time domain features for each segment are calculated and assembled in a feature matrix. There are 200 segments for each fault and we have taken 9 time domain features. The time domain features are maximum, minimum, mean value, standard deviation, root mean square value (RMS), skewness, kurtosis, crest factor, and form factor. Thus our feature matrix is of size (2000×9). One column containing fault type is also added to the feature matrix. Thus final feature matrix is of size (2000×10).

Before applying SVM, the data are first separated into a training set and a test set. The test set contains 50 rows of fault matrix chosen for each fault type. Thus its size is (500×10). The rest are taken as training set.

SVM is applied to training set data and best parameters are chosen by cross validation. The best parameters are then applied to test set data to predict final classification result. We will use R to implement SVM. We will use Python to plot the confusion matrix.

```
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(e1071)
data_time = read.csv('./feature_matrix/48k/feature_time_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_time[-index,]
test_data = data_time[index,]

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

We apply cross-validation over a different set of parameters to obtain best set of parameters. This cross-validation is done by the 'tune' command and the parameters considered are the cost and gamma values as mentioned in the codes. Radial basis is used. The command 'svm_tune$best.model' is the best model obtained from cross validation. This model is used in later lines.

```
set.seed(11)
svm_tune = tune(svm,train_data[,-dim(train_data)[2]],
                train_data[,dim(train_data)[2]],kernel = 'radial',
                ranges = list(cost = c(1,10,50,100,200,300),
                              gamma = c(0.05,0.1,0.5,1,5)))
pred_train = predict(svm_tune$best.model,train_data[,-dim(train_data)[2]])
pred_test = predict(svm_tune$best.model,test_data[,-dim(train_data)[2]])
# Confusion matrix
train_confu = table(train_data[,dim(train_data)[2]],pred_train)
test_confu = table(test_data[,dim(train_data)[2]],pred_test)
```

Finally, we will use python's seaborn package to visualize confusion matrix for both training and test data. RStudio makes it convenient to run both R and Python scripts simultaneously. RStudio is great!

```
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, cmap = "Blues",fmt = "d",
xticklabels=fault_type, yticklabels=fault_type)
```

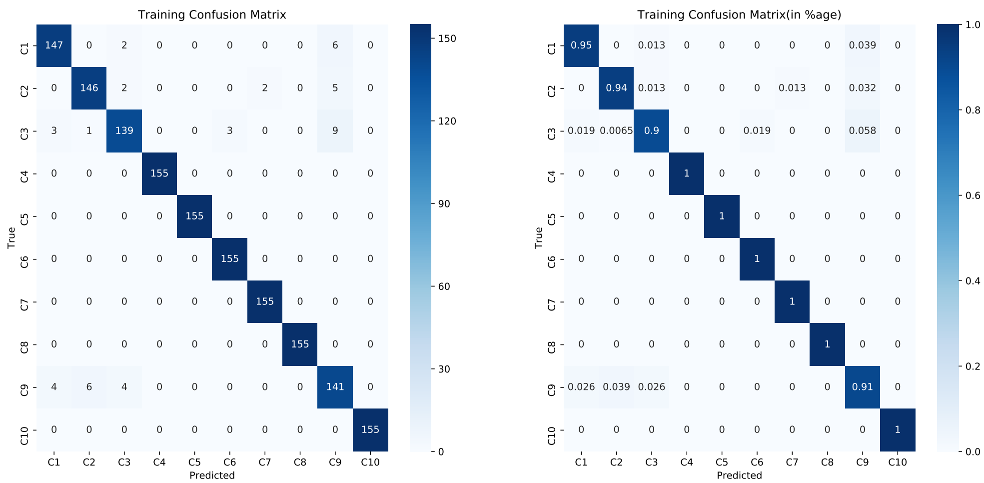
```
## <matplotlib.axes._subplots.AxesSubplot object at 0x0000000024F7A438>
```

```
plt.title('Training Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
```

```
plt.subplot(122)
sns.heatmap(r.train_confu/155, annot= True, cmap = "Blues",
xticklabels=fault_type, yticklabels=fault_type)
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x0000000025FD2320>
```

```
plt.title('Training Confusion Matrix(in %age)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.subplot(122)
plt.show()
```



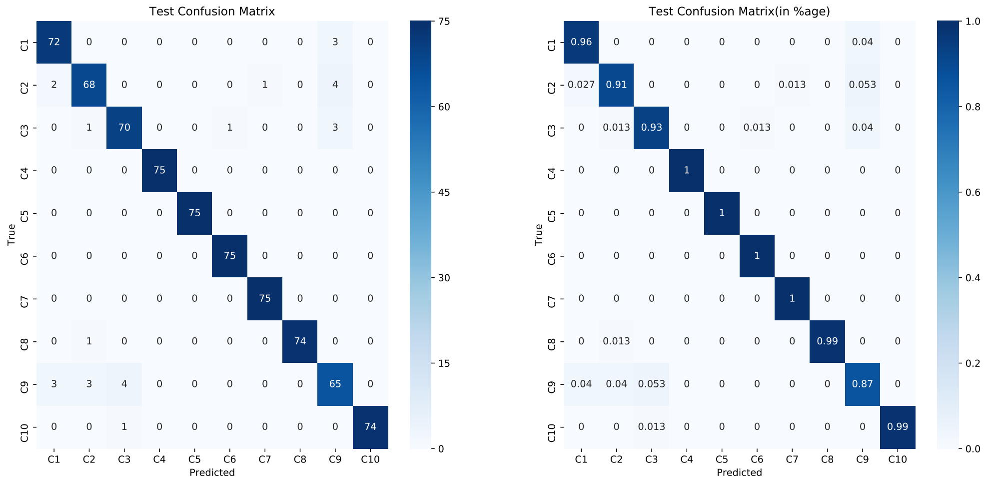

```
plt.figure(2,figsize=(18,8))
plt.subplot(121)
sns.heatmap(r.test_confu, annot = True, cmap = "Blues",
xticklabels=fault_type, yticklabels=fault_type)
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x0000000015FC4CF8>
```

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

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000000260372E8>
```

```
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: 96.4000"
```

The overall test accuracy is 96.4% which is pretty satisfactory considering the fact that we are only taking time domain features. We will show in the next post that accuracy improves even further when wavelet features are used. Check this page for other methods.

Last modified: 12$^{\text{th}}$ June, 2019