Multiclass classification using SVM on wavelet packet entropy features

Biswajit Sahoo

(Click this link to read the previous post on wavlelet packet energy. In this post we have used wavelet packet entropy)

This is the third post on multiclass classification using SVM. In the last two articles, we had used time domain features and wavelet packet energy features as input to SVM and seen its results. Here, we will apply SVM on wavelet packet entropy features. We will see that this result is almost equal to the result of using wavelet packet energy. We will again use Case Western Reserve University Bearnig data set for our multiclass classification problem.

Description of data set

(We will use the same data set and the description is same as previous case. We will repeat the same thing here for the sake of completeness and to make it independent of previous one.)

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 at a sampling frequency of 12 kHz and some other are collected at 48 kHz. In this study, we will only consider data acquired at 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 will consider all the data with 1 hp external load.

There are 10 classes 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.

Wavelet analysis is a signal processing technique that gives us time-frequency representation. The inner workings of wavelets are different than traditional time-frequency methods (for example, spectrograms). We will not go into the details of wavelets here (It is saved for a later post.). Wavelet packets are a way of segregating a signal into different frequency bands. A pictorial representation will make the difference between wavelets and wavelet packets clear. (Note on notation: S-Signal, H- Low pass filter, G- High pass filter, $\downarrow 2$ - Downsampling by factor of 2, A1- Approximation coefficients corresponding to low frequency, D1- Detail coefficients corresponding to high frequency)

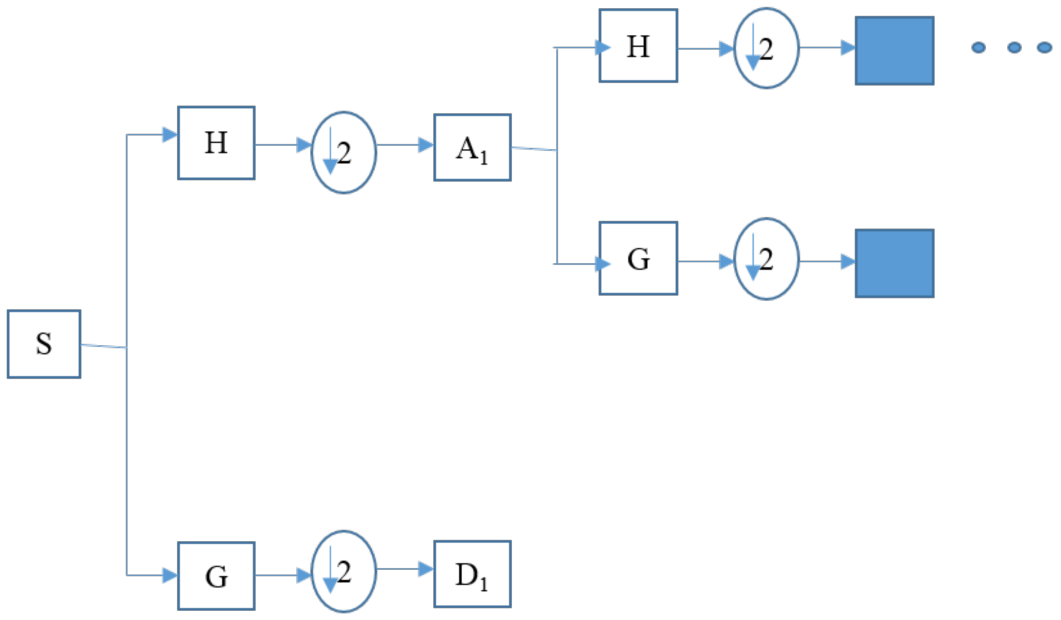


Figure 1: Wavelet decomposition

In wavelet packet transform, after three stages of transformation we get 8 nodes. From each node coefficients we reconstruct the signal and obtain its entropy. Thus we get 8 entropy features and because these entropies are calculated from wavelet packets, we call these wavelet packet entropy features. There are many mother wavelets that can be used (for example, Haar, Sym8, Daubechies, etc.). We have used 'Sym8' as a shrinked version of it can approximate impulses well.

Here is how we form feature matrix. 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 8 wavelet packet features for each segment are calculated and assembled in a feature matrix. There are 230 segments for each fault and we have taken 8 wavelet packets. (We could also have considered 16 wavelet packet thus getting 16 wavelet packet features. But we have stopped at 8 as is usually done in research.) Thus our feature matrix is of size (2300×8). One column containing fault type is also added to the feature matrix. Thus final feature matrix is of size (2300×9).

Before applying SVM, the data are first separated into a training set and a test set. The test set contains 75 rows of fault matrix chosen for each fault type. Thus its size is (750×9). The rest are taken as training set.

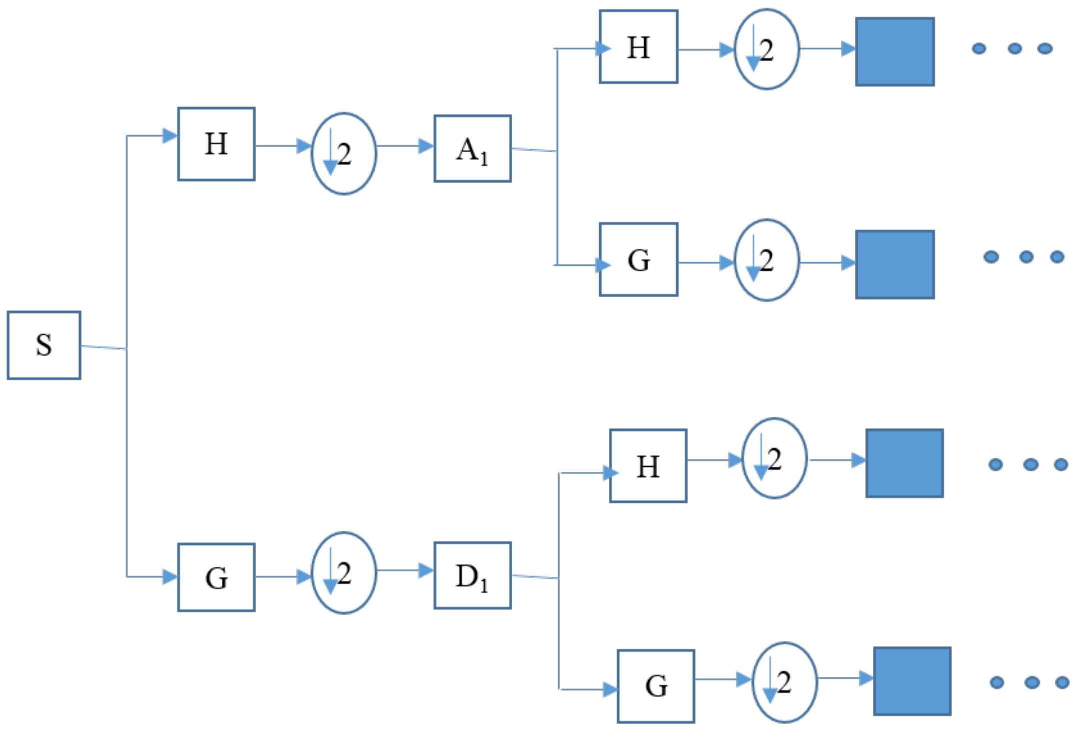


Figure 2: Wavelet packet decomposition

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 have not printed the intermediate results. Readers can easily print those as per requirement. We will use R to implement SVM. To plot confusion matrix, we will use Python.

Codes

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

How to get data?

Readers can download the `.csv` file used in this notebook from here. Another convenient way is to download the whole repository and run the downloaded notebooks.

```
library(e1071)
data_wav_entropy = read.csv("./data/feature_wav_ent8_shan_48k_2048_load_1.csv", header = T)
# Change the above line to include your folder that contains data
set.seed(312)
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_entropy[-index,]
test_data = data_wav_entropy[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(10,50,100,500,2000),
                              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!

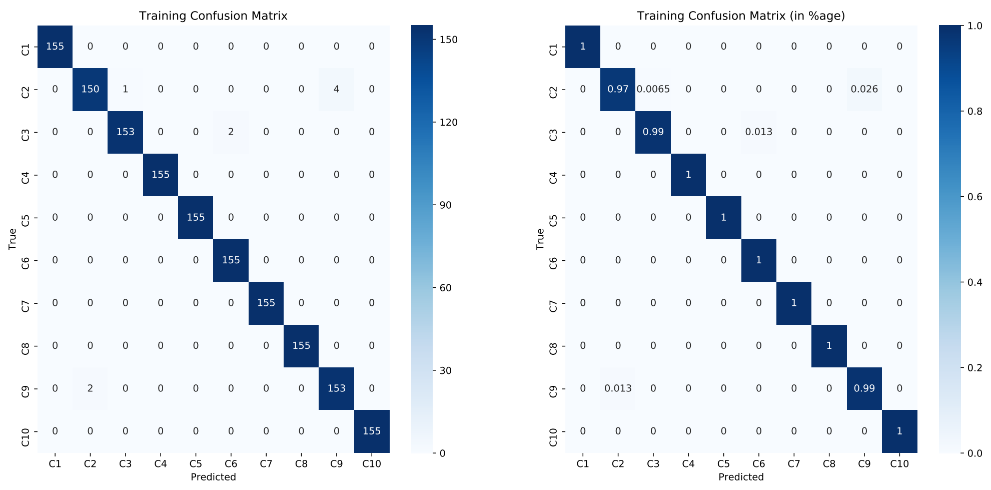
```
import seaborn as sns
import matplotlib.pyplot as plt
fault_type = ['C1','C2','C3','C4','C5','C6','C7','C8','C9','C10']
plt.figure(1,figsize=(18,8))
plt.subplot(121)
sns.heatmap(r.train_confu, annot= True,fmt = "d",
xticklabels=fault_type, yticklabels=fault_type, cmap = "Blues")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x000000001A6B2488>
```

```
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 0x0000000019DF0F08>
```

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



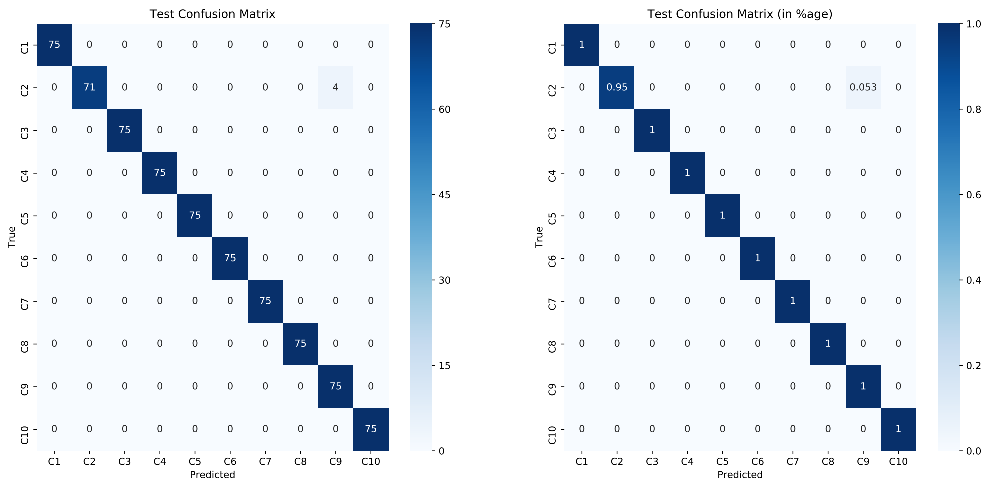
```
plt.figure(2,figsize=(18,8))
plt.subplot(121)
sns.heatmap(r.test_confu, annot = True,
xticklabels=fault_type, yticklabels=fault_type, cmap = "Blues")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x000000002B9AFDC8>
```

```
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 0x000000002B8CD148>
```

```
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: 99.4667"
```

Now the overall test accuracy is 99.4% which is just one decimal point less than the accuracy obtained by wavelet packet energy features. So wavelet packet features have proved to be effective for classifying faults using SVM. In future we will apply other techniques to CWRU data and compare their performance with this. Check this page for further details.

Note:

- More details about wavelet packet features and the ways (with codes) to compute it will appear in a future post. For the time being, readers can download and use the feature matrix already created by us. Check this page for latest updates.

```
sessionInfo()
```

```
## R version 3.6.2 (2019-12-12)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 7 x64 (build 7601) Service Pack 1
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
```

```
##
## other attached packages:
## [1] e1071_1.7-2     reticulate_1.14
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.3      class_7.3-15    digest_0.6.23   rappdirs_0.3.1
##  [5] jsonlite_1.6.1  magrittr_1.5    evaluate_0.14   rlang_0.4.4
##  [9] stringi_1.4.5   rmarkdown_2.1   tools_3.6.2     stringr_1.4.0
## [13] xfun_0.12       yaml_2.2.0      compiler_3.6.2  htmltools_0.4.0
## [17] knitr_1.27
```

Last updated: 14^{th} February, 2020