Predictive Maintenance-Fault classification

Nagdev Amruthnath
Tyson Foods
25 PUBLICATIONS **202** CITATIONS

Predictive Maintenance - Fault classification

February 27, 2019

1 Introduction to Predictive Maintenance

1.1 Fault Classification using Supervised learning

Author Nagdev Amruthnath Date: 1/10/2019

Citation Info If you are using this for your research, please use the following for citation.
Amruthnath, Nagdev, and Tarun Gupta. "A research study on unsupervised machine learning algorithms for early fault detection in predictive maintenance." In 2018 5th International Conference on Industrial Engineering and Applications (ICIEA), pp. 355-361. IEEE, 2018.

Disclaimer This is a tutorial for performing fault detection using machine learning. You this code at your own risk. I do not gurantee that this would work as shown below. If you have any suggestions please branch this project.

1.2 Introduction

This is the first of four part demostration series of using machine learning for predictive maintenance.

The area of predictive maintenance has taken a lot of prominence in the last couple of years due to various reasons. With new algorithms and methodologies growing across different learning methods, it has remained a challenge for industries to adopt which method is fit, robust and provide most accurate detection. One the most common learning approaches used today for fault diagnosis is supervised learning. This is wholly based on the predictor variable and response variable. In this tutorial, we will be looking into some of the common supervised learning models such as SVM, random forest, k-nearest neighbour and H2O's AutoML model.

1.3 Load libraries

```
In [6]: options(warn=-1)

        # load libraries
        library(mdatools) #mdatools version 0.9.1
        library(caret)
        library(foreach)
        library(dplyr)
        library(h2o)
library(doParallel, verbose = F)
library(ModelMetrics, verbose = F)
```

1.4 Load data

Here we are using data from a bench press. There are total of four different states in this machine and they are split into four different csv files. We need to load the data first. In the data time represents the time between samples, ax is the acceleration on x axis, ay is the acceleration on y axis, az is the acceleration on z axis and at is the G's. The data was collected at sample rate of 100hz.

Four different states of the machine were collected
1. Nothing attached to drill press 2. Wooden base attached to drill press 3. Imbalance created by adding weight to one end of wooden base 4. Imbalacne created by adding weight to two ends of wooden base.

```
In [7]: #setwd("/Experiment")
        #read csv files
        file1 = read.csv("dry run.csv", sep=",", header =T)
        file2 = read.csv("base.csv", sep=",", header =T)
        file3 = read.csv("imbalance 1.csv", sep=",", header =T)
        file4 = read.csv("imbalance 2.csv", sep=",", header =T)

        #Add labels to data
        file1$y = 1
        file2$y = 2
        file3$y = 3
        file4$y = 4

        #view top rows of data
        head(file1)
```

time	ax	ay	az	aT	y
0.002	-0.3246	0.2748	0.1502	0.451	1
0.009	0.6020	-0.1900	-0.3227	0.709	1
0.019	0.9787	0.3258	0.0124	1.032	1
0.027	0.6141	-0.4179	0.0471	0.744	1
0.038	-0.3218	-0.6389	-0.4259	0.833	1
0.047	-0.3607	0.1332	-0.1291	0.406	1

We can look at the summary of each file using summary function in R. Below, we can observe that 66 seconds long data is available. We also have min, max and mean for each of the variables.

```
In [8]: # summary of each file
        summary(file1)
```

```
      time              ax                  ay                   az
 Min.   : 0.002   Min.   :-2.11880   Min.   :-2.143600   Min.   :-4.1744
 1st Qu.:16.507   1st Qu.:-0.41478   1st Qu.:-0.625250   1st Qu.:-0.7359
 Median :33.044   Median : 0.02960   Median :-0.022050   Median :-0.1468
 Mean   :33.037   Mean   : 0.01233   Mean   : 0.008697   Mean   :-0.1021
 3rd Qu.:49.535   3rd Qu.: 0.46003   3rd Qu.: 0.641700   3rd Qu.: 0.4298  
 Max.   :66.033   Max.   : 2.09620   Max.   : 2.003000   Max.   : 4.9466  
       aT               y    
 Min.   :0.032   Min.   :1  
 1st Qu.:0.848   1st Qu.:1  
 Median :1.169   Median :1  
 Mean   :1.277   Mean   :1  
 3rd Qu.:1.579   3rd Qu.:1  
 Max.   :5.013   Max.   :1  
```

1.5 Data Aggregration and feature extraction

Here, the data is aggregated by 1 minute and features are extracted. Features are extracted to reduce the dimension of the data and only storing the representation of the data.

```
In [9]: file1$group = as.factor(round(file1$time))
        file2$group = as.factor(round(file2$time))
        file3$group = as.factor(round(file3$time))
        file4$group = as.factor(round(file4$time))
        #(file1,20)

        #list of all files
        files = list(file1, file2, file3, file4)

        #loop through all files and combine
        features = NULL
        for (i in 1:4){
        res = files[[i]] %>%
            group_by(group) %>%
            summarize(ax_mean = mean(ax),
                      ax_sd = sd(ax),
                      ax_min = min(ax),
                      ax_max = max(ax),
                      ax_median = median(ax),
                      ay_mean = mean(ay),
                      ay_sd = sd(ay),
                      ay_min = min(ay),
                      ay_may = max(ay),
                      ay_median = median(ay),
                      az_mean = mean(az),
                      az_sd = sd(az),
                      az_min = min(az),
                      az_maz = max(az),
                      az_median = median(az),
                      aT_mean = mean(aT),
                      aT_sd = sd(aT),
                      aT_min = min(aT),
                      aT_maT = max(aT),
                      aT_median = median(aT),
                      y = mean(y)
                     )
          features = rbind(features, res)
      }

      #view all features
      features$y = as.factor(features$y)
      head(features)
```

group	ax_mean	ax_sd	ax_min	ax_max	ax_median	ay_mean	ay_sd	ay_min
0	-0.038164706	0.6594686	-1.2587	1.3821	-0.0955	-0.0682627451	0.7506785	-1.3892
1	-0.005806122	0.6325808	-1.6194	1.1943	-0.0015	0.0037908163	0.7819044	-1.5625
2	0.069845455	0.6665500	-1.4554	1.4667	0.1070	0.0744333333	0.8022922	-1.4800
3	0.011552525	0.5511310	-1.9254	1.2034	0.0675	0.0008262626	0.7894209	-2.0042
4	0.046688119	0.6426574	-1.7805	1.4837	0.0836	-0.0177594059	0.7510811	-1.6629
5	0.006678788	0.5780957	-1.4719	1.4355	0.0536	0.0013626263	0.7812245	-1.6293

1.6 Create sample size for training the model

From the information, we know that we have four states in the data. Based on this information, the data is split into train and test samples. The train set is used to build the model and test set is used to validate the model. The ratio between train and test is 80:20. You can adjust this based on type of data. The below table shows the number of observations for each group.

Note: It is adviced to have atleast 30 samples for each group.

```
In [10]: table(features$y)
```

```
  1   2   3   4
 67 109  93  93
```

From the above results, we can observe that there are atleast 30 samples for each group. Now, we can used this data to split into train and test set.

```
In [11]: #create samples of 80:20 ratio
         sample = sample(nrow(features) , nrow(features)* 0.8)
         train = features[sample,]
         test = features[-sample,]
```

1.7 Supervised Fault Classification

1.7.1 Fault Classification using Random Forest

Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because it's simplicity and the fact that it can be used for both classification and regression tasks.

In this post, you are going to learn, how the random forest algorithm works and several other important things about it [3].

More about Random forest can be learnt here.

https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd

```
In [12]: #If you don't want parallel computing, comment bellow lines of code.
         #create parallel clusters for 16 cores
         cl <- makeCluster(16)
         registerDoParallel(cl)

         #set training partameters
         fitControl = trainControl(method = "repeatedcv",
                                   number = 10,
                                   ## repeated ten times
                                   repeats = 20)

         rf.model = train(y ~ ., train,
                          method = "parRF",
                          allowParallel = TRUE,
                          metric ="Kappa",
                          trControl = fitControl,
                          verbose = FALSE
                          #rfeControl=control
                         )

         #summary of trained model
         rf.model

         #close all clusters
         stopCluster(cl)
```

```
Parallel Random Forest

289 samples
 21 predictor
  4 classes: '1', '2', '3', '4'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 20 times)
Summary of sample sizes: 260, 259, 260, 260, 260, 260, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
    2   0.9302873  0.9060006
   65   0.9795237  0.9725126
  128   0.9767511  0.9687818

Kappa was used to select the optimal model using the largest value.
```

```
The final value used for the model was mtry = 65.
```

From the training results, we can note that 97% accuracy and 96% kappa value has been achieved. This indicates a very good model. Next, we need to validate based on test data set and compute the validation accuracy.

```
In [13]: # Used the model to perform prediction
         prediction = data.frame(pred=predict.train(rf.model, test))

         #create a confusion matrix
         caret::confusionMatrix(test$y, prediction$pred)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3  4
         1 12  0  0  0
         2  0 20  0  0
         3  0  0 21  0
         4  0  0  1 19

Overall Statistics

               Accuracy : 0.9863
                 95% CI : (0.926, 0.9997)
    No Information Rate : 0.3014
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9815
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            1.0000    1.000   0.9545   1.0000
Specificity            1.0000    1.000   1.0000   0.9815
Pos Pred Value         1.0000    1.000   1.0000   0.9500
Neg Pred Value         1.0000    1.000   0.9808   1.0000
Prevalence             0.1644    0.274   0.3014   0.2603
Detection Rate         0.1644    0.274   0.2877   0.2603
Detection Prevalence   0.1644    0.274   0.2877   0.2740
Balanced Accuracy      1.0000    1.000   0.9773   0.9907
```

From the above results, we can observe that the following model achived highest accuracy. Also, no information rate is less than accuracy indicating that the model is reliable. Also, kappa value is very high indicating similar inference.

1.7.2 Fault Classification using Support Vector Machine (SVM)

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side [4]

An indetail tutorial regarding SVM can be found here. https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

```
In [14]: #If you don't want parallel computing, comment bellow lines of code.
         #create parallel clusters for 16 cores
         cl <- makeCluster(16)
         registerDoParallel(cl)

         #set training partameters
         fitControl = trainControl(method = "repeatedcv",
                                   number = 10,
                                   ## repeated ten times
                                   repeats = 20)

         rf.model = train(y ~ ., train,
                         method = "svmLinear",
                         allowParallel = TRUE,
                         metric ="Kappa",
                         trControl = fitControl,
                         verbose = FALSE
                        )

         #summary of trained model
         rf.model

         #close all clusters
         stopCluster(cl)
```

```
Support Vector Machines with Linear Kernel

289 samples
 21 predictor
  4 classes: '1', '2', '3', '4'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 20 times)
Summary of sample sizes: 260, 261, 260, 260, 261, 260, ...
Resampling results:

  Accuracy   Kappa
  0.9580475  0.94358

Tuning parameter 'C' was held constant at a value of 1
```

From the above results we can observe that accuracy and kappa values have decreased. But, this decrease is very minor and negligible. Next we can validate with test data

```
In [15]: # Used the model to perform prediction
         prediction = data.frame(pred=predict.train(rf.model, test))

         #create a confusion matrix
         caret::confusionMatrix(test$y, prediction$pred)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3  4
         1 12  0  0  0
         2  0 20  0  0
         3  0  0 19  2
         4  0  0  0 20

Overall Statistics

               Accuracy : 0.9726
                 95% CI : (0.9045, 0.9967)
    No Information Rate : 0.3014
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.963
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            1.0000    1.000   1.0000   0.9091
Specificity            1.0000    1.000   0.9630   1.0000
Pos Pred Value         1.0000    1.000   0.9048   1.0000
Neg Pred Value         1.0000    1.000   1.0000   0.9623
Prevalence             0.1644    0.274   0.2603   0.3014
Detection Rate         0.1644    0.274   0.2603   0.2740
Detection Prevalence   0.1644    0.274   0.2877   0.2740
Balanced Accuracy      1.0000    1.000   0.9815   0.9545
```

From the above validation results, we can conclude that the results have marginally decreased compared to random forest model. Accuracy is much higher than NIR which indicates that this is still a reliable model.

1.7.3 Fault Classification using K-nearest Neighbour (KNN)

KNN is one of the simplest supervised learning method. This model can be used for both classification and regression models. One of the drawbacks of this model is, this is on the fly training model. This is a time consuming model. As the amount of data increases, the training time increases.

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique [5].

```
In [17]: #create parallel clusters for 16 cores
         cl <- makeCluster(16)
         registerDoParallel(cl)
         #set training partameters
         fitControl = trainControl(method = "repeatedcv",
                                    ## repeated ten times
                                    repeats = 3)

         rf.model = train(y ~ ., train,
                          method = "knn",
                          trControl = fitControl,
                          preProcess = c("center","scale")
                         )

         #summary of trained model
         rf.model


         #close all clusters
         stopCluster(cl)
```

```
k-Nearest Neighbors

289 samples
 21 predictor
  4 classes: '1', '2', '3', '4'

Pre-processing: centered (128), scaled (128)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 261, 260, 260, 260, 261, 259, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.2919359  0.06488064
  7  0.7431069  0.65660220
  9  0.8702394  0.82535956

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 9.
```

From the train results we can note that the accuracy of KNN is much lower than RandomForest and SVM. Next, we need to test the accuracy of validation data.

```
In [18]: # Used the model to perform prediction
         prediction = data.frame(pred=predict.train(rf.model, test))

         #create a confusion matrix
         caret::confusionMatrix(test$y, prediction$pred)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3  4
         1 12  0  0  0
         2  1 19  0  0
         3  1  0 15  5
         4  0  0  0 20

Overall Statistics

               Accuracy : 0.9041
                 95% CI : (0.8124, 0.9606)
    No Information Rate : 0.3425
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8712
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.8571   1.0000   1.0000   0.8000
Specificity            1.0000   0.9815   0.8966   1.0000
Pos Pred Value         1.0000   0.9500   0.7143   1.0000
Neg Pred Value         0.9672   1.0000   1.0000   0.9057
Prevalence             0.1918   0.2603   0.2055   0.3425
Detection Rate         0.1644   0.2603   0.2055   0.2740
Detection Prevalence   0.1644   0.2740   0.2877   0.2740
Balanced Accuracy      0.9286   0.9907   0.9483   0.9000
```

From the confusion matrix we can observe that the accuracy and kappa value is significantly lower than the values compared to random forest and SVM model.

1.7.4 Fault Classification using AutoML

The H2O AutoML interface is designed to have as few parameters as possible so that all the user needs to do is point to their dataset, identify the response column and optionally specify a time constraint or limit on the number of total models trained [6].

The below model will be trained for 30 seconds for this tutorial. This can be scaled up or down based on needs.

```
In [19]: #initialize H2O
         h2o.init()

         #change to h2o dataframe.
         trainAML = as.h2o(train)
         testAML = as.h2o(test)

         # Identify predictors and response
         y = "y"
         x <- setdiff(names(train), y)

         # For binary classification, response should be a factor
         trainAML[,y] = as.factor(trainAML[,y])
         testAML[,y] = as.factor(testAML[,y])

         #train model using AutoML
         aml <- h2o.automl(y = y, training_frame = trainAML, leaderboard_frame = testAML,
             max_runtime_secs = 10)

         # View the AutoML Leaderboard
         lb <- aml@leaderboard
         lb
```

```
 Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         2 days 43 minutes
    H2O cluster timezone:       America/New_York
    H2O data parsing timezone:  UTC
    H2O cluster version:        3.20.0.2
    H2O cluster version age:    8 months and 11 days !!!
    H2O cluster name:           H2O_started_from_R_aanamruthn_tdy557
    H2O cluster total nodes:    1
    H2O cluster total memory:   7.48 GB
    H2O cluster total cores:    16
    H2O cluster allowed cores:  16
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    H2O API Extensions:         Algos, AutoML, Core V3, Core V4
    R Version:                  R version 3.5.1 (2018-07-02)
```

```
  |======================================================================| 100%
  |======================================================================| 100%
  |======================================================================| 100%


                                               model_id mean_per_class_error
1             GBM_grid_0_AutoML_20190227_152539_model_0           0.01250000
2                          XRT_0_AutoML_20190227_152539           0.02380952
3                          DRF_0_AutoML_20190227_152539           0.03630952
4             GLM_grid_0_AutoML_20190227_152539_model_0           0.67857143
5 StackedEnsemble_BestOfFamily_0_AutoML_20190227_152539           0.75000000
6    StackedEnsemble_AllModels_0_AutoML_20190227_152539           0.75000000
     logloss      rmse        mse
1 0.06791937 0.1136454 0.01291529
2 0.07421694 0.1385458 0.01919493
3 0.10305847 0.1574621 0.02479430
4 1.30128990 0.7245946 0.52503733
5 1.37134105 0.7439822 0.55350945
6 1.37134105 0.7439822 0.55350945

[6 rows x 5 columns]
```

```
In [20]: pred = h2o.predict(aml@leader, testAML)
         preds = as.data.frame(pred)
         #create a confusion matrix
         caret::confusionMatrix(test$y, preds$predict)
```

```
  |======================================================================| 100%


Confusion Matrix and Statistics

          Reference
Prediction  1  2  3  4
         1 12  0  0  0
         2  0 20  0  0
         3  0  0 21  0
         4  0  0  1 19

Overall Statistics

               Accuracy : 0.9863
                 95% CI : (0.926, 0.9997)
    No Information Rate : 0.3014
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9815
 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            1.0000    1.000   0.9545   1.0000
Specificity            1.0000    1.000   1.0000   0.9815
Pos Pred Value         1.0000    1.000   1.0000   0.9500
Neg Pred Value         1.0000    1.000   0.9808   1.0000
Prevalence             0.1644    0.274   0.3014   0.2603
Detection Rate         0.1644    0.274   0.2877   0.2603
Detection Prevalence   0.1644    0.274   0.2877   0.2740
Balanced Accuracy      1.0000    1.000   0.9773   0.9907
```

The model used during AutoML process is GBM. This model provided the highest accuracy during training. From the validation results, we can observe that the accuracy is much higher than NIR and Kappa is also much higher compared to all the other models in this tutorial.

References [1] Amruthnath, Nagdev, and Tarun Gupta. "A research study on unsupervised machine learning algorithms for early fault detection in predictive maintenance." In 2018 5th International Conference on Industrial Engineering and Applications (ICIEA), pp. 355-361. IEEE, 2018.

[2] Amruthnath, Nagdev, and Tarun Gupta. "Fault class prediction in unsupervised learning using model-based clustering approach." In Information and Computer Technologies (ICICT), 2018 International Conference on, pp. 5-12. IEEE, 2018.

[3] Niklas Donges, "The Random Forest Algorithm", https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd.

[4] Savan Patel, "Chapter 2 : SVM (Support Vector Machine)—Theory", https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

[5] "K Nearest Neighbors - Classification", https://www.saedsayad.com/k_nearest_neighbors.htm

[6] "AutoML: Automatic Machine Learning", http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html