Accuracy Improvement

Rischan Mafrur

Monday, December 22, 2014

As I told in previous documentation about our analysis result, we only get 71 % accuracy and its only using time domain features because frequency domain features does not have good performance.

In this document, I tried to improve our result with best features selection methods and tried to using svm tune to know the best svm parameters for our dataset. Features selection method that we used in this project is SFS (sequential forward selection) and SFFS (sequential forward floating selection) but in this project I prefer to only use SFFS because this algorithm perform better than another one.

To use SFFS algorithem in R, we need to install and load "mlr" library

```
#install.packages("mlr")
library(mlr)

library(e1071)

library(caret)
```

In this documents similar with previous, we also divide to three experiments: Using all features (combine time domain features and frequency domain features), only using time domain features, and the third only using frequecy domain features.

*Please read comment inside the code.

All Features Dataset

Set working directory and load the dataset

```
setwd("D:/Dropbox/LAB/COURSE/3/ubi/data") #set working directory

features <- read.csv("all_features.csv",header = TRUE) #load all of dataset f
rom all features
```

SFFS Algorithm

```
######### SFFS Algorithm
f.task = makeClassifTask(data = features, target = "label")
rdesc = makeResampleDesc("CV")
ctrl = makeFeatSelControlSequential(method = "sffs", maxit = NA)
res = selectFeatures("classif.rpart", f.task, rdesc, control = ctrl)
```

```
## Loading required package: rpart
## [FeatSel] Started selecting features for learner 'classif.rpart'
## With control class: FeatSelControlSequential
## Imputation value: 1
## [FeatSel] 1: 0 bits: mmce.test.mean=0.741
## [FeatSel] 2: 1 bits: mmce.test.mean=0.654
## [FeatSel] 2: 1 bits: mmce.test.mean=0.609
## [FeatSel] 2: 1 bits: mmce.test.mean=0.65

-----------------------
## [FeatSel] 15: 5 bits: mmce.test.mean=0.38
## [FeatSel] 15: 5 bits: mmce.test.mean=0.289
## [FeatSel] 15: 5 bits: mmce.test.mean=0.316
## [FeatSel] 15: 5 bits: mmce.test.mean=0.327
## [FeatSel] 15: 5 bits: mmce.test.mean=0.363
## [FeatSel] 15: 5 bits: mmce.test.mean= 0.3
## [FeatSel] Result: 7 bits : mmce.test.mean=0.278
```

```
analyzeFeatSelResult(res)
```

```
## Features         : 7
## Performance      : mmce.test.mean=0.278
## MeanX, MeanY, MaxY, MinY, MeanZ, SdZ, MinM
##
## Path to optimum:
## - Features:    0  Init   :                      Perf = 0.74142  Diff: NA
 *
## - Features:    1  Add    : MinY                 Perf = 0.49032  Diff: 0.2
511  *
## - Features:    2  Add    : MeanX                Perf = 0.40382  Diff: 0.0
86506  *
## - Features:    3  Add    : MeanY                Perf = 0.36653  Diff: 0.0
37284  *
## - Features:    4  Add    : SdZ                  Perf = 0.33021  Diff: 0.0
36323  *
## - Features:    5  Add    : MeanZ                Perf = 0.30035  Diff: 0.0
29853  *
## - Features:    6  Add    : MinM                 Perf = 0.28395  Diff: 0.0
16406  *
##
## Stopped, because no improving feature was found.
```

```
######### SFFS Algorithm End Here
```

Start to Applying SVM Classification

```
# Based on SFFS Algorithm the best features which has good performance are "M
eanX", "AbsX", "MeanY", "MinY", "MeanZ", "SdZ". So in this experiment we only
 use those of features.
new_f <- subset(features, select=c("MeanX", "AbsX", "MeanY", "MinY", "MeanZ",
 "SdZ","label"))
```

```r
#Function for splitting dataset.
splitdf <- function(dataframe, seed=NULL) {
  if (!is.null(seed)) set.seed(seed)
  index <- 1:nrow(dataframe)
  trainindex <- sample(index, trunc(length(index)*(90/100)))
  trainset <- dataframe[trainindex, ]
  testset <- dataframe[-trainindex, ]
  list(trainset=trainset,testset=testset)
}

splits <- splitdf(new_f, seed=808)
str(splits)
```

```
## List of 2
##  $ trainset:'data.frame':    1809 obs. of  7 variables:
##   ..$ MeanX: num [1:1809] 0.388 0.677 1.975 -0.295 -0.628 ...
##   ..$ AbsX : num [1:1809] 0.1384 0.0664 0.4515 1.2965 1.0418 ...
##   ..$ MeanY: num [1:1809] -9.71 -8.12 -9.84 -10.33 -9.98 ...
##   ..$ MinY : num [1:1809] -10.15 -9.01 -12.96 -14.07 -12.38 ...
##   ..$ MeanZ: num [1:1809] 1.92 3.927 -0.356 -1.271 -1.235 ...
##   ..$ SdZ  : num [1:1809] 0.65 1.436 3.012 3.276 0.893 ...
##   ..$ label: Factor w/ 4 levels "agung","alvin",..: 4 1 2 3 4 2 1 4 2 2 ..
.
##  $ testset :'data.frame':    202 obs. of  7 variables:
##   ..$ MeanX: num [1:202] 0.5974 0.0769 -0.2685 -0.3494 1.4408 ...
##   ..$ AbsX : num [1:202] 1.587 2.64 0.178 1.86 3.158 ...
##   ..$ MeanY: num [1:202] -10.95 -9.52 -9.49 -10.71 -12.04 ...
##   ..$ MinY : num [1:202] -17.86 -14.8 -9.91 -18.79 -17.55 ...
##   ..$ MeanZ: num [1:202] 1.035 0.952 3.859 1.169 2.61 ...
##   ..$ SdZ  : num [1:202] 4.45 7.22 1.19 1.69 7.22 ...
##   ..$ label: Factor w/ 4 levels "agung","alvin",..: 1 1 1 1 1 1 1 1 1 1 ..
.
```

```r
lapply(splits,nrow)
```

```
## $trainset
## [1] 1809
##
## $testset
## [1] 202
```

```r
lapply(splits,head)
```

```
## $trainset
##           MeanX       AbsX      MeanY       MinY      MeanZ       SdZ
## 1830  0.3881605 0.13839924  -9.713926 -10.146953  1.9198548 0.6497238
## 390   0.6772714 0.06637231  -8.122215  -9.013465  3.9272541 1.4361473
## 730   1.9745577 0.45150434  -9.838797 -12.957370 -0.3558127 3.0116910
```

```
## 1552 -0.2953630 1.29653156 -10.325177 -14.068901 -1.2712022 3.2755954
## 1976 -0.6275057 1.04180666  -9.979438 -12.380973 -1.2346527 0.8928233
## 811  -0.1500456 0.54080810 -10.087610 -11.291659  0.7369852 0.2269612
##        label
## 1830 rischan
## 390    agung
## 730    alvin
## 1552     gde
## 1976 rischan
## 811    alvin
##
## $testset
##          MeanX      AbsX      MeanY      MinY      MeanZ      SdZ label
## 5   0.59742959 1.5867704 -10.954408 -17.85851  1.0352185 4.448004 agung
## 7   0.07686185 2.6395196  -9.519404 -14.80353  0.9521123 7.219119 agung
## 17 -0.26854840 0.1779955  -9.490250  -9.90878  3.8594697 1.186131 agung
## 30 -0.34944954 1.8598252 -10.710136 -18.79459  1.1687857 1.686225 agung
## 34  1.44083419 3.1577843 -12.042180 -17.55239  2.6103309 7.215945 agung
## 42  1.27533278 0.6075398  -9.865321 -17.50970 -0.1746398 6.212753 agung
```

```
training <- splits$trainset
testing <- splits$testset

#Starting SVM Classification
x <- subset(training, select=-label)
y <- training$label

#Tunning SVM Parameters, this function is for looking the best SVM parameters
 for our dataset.
svm_tune <- tune(svm, train.x=x, train.y=y,
                 kernel="radial", ranges=list(cost=10^(-1:2), gamma=c(.5,1,2)
))

summary(svm_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.5
##
## - best performance: 0.2073082
##
## - Detailed performance results:
##    cost gamma     error dispersion
## 1   0.1   0.5 0.3388735 0.02761753
```

```
## 2    1.0   0.5 0.2227655 0.02448603
## 3   10.0   0.5 0.2073082 0.02211028
## 4  100.0   0.5 0.2260958 0.02174847
## 5    0.1   1.0 0.3482781 0.02892765
## 6    1.0   1.0 0.2150430 0.02357730
## 7   10.0   1.0 0.2128207 0.02386454
## 8  100.0   1.0 0.2277502 0.02383854
## 9    0.1   2.0 0.3764579 0.03457708
## 10   1.0   2.0 0.2205709 0.03168064
## 11  10.0   2.0 0.2310743 0.02622156
## 12 100.0   2.0 0.2526335 0.02727779
```

```
#creating training model
svm_model <- svm(x,y,kernel="radial", cost=10, gamma=0.5)
summary(svm_model)
```

```
##
## Call:
## svm.default(x = x, y = y, kernel = "radial", gamma = 0.5, cost = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.5
##
## Number of Support Vectors:  1064
##
##  ( 270 227 236 331 )
##
##
## Number of Classes:  4
##
## Levels:
##  agung alvin gde rischan
```

```
pred <- predict(svm_model,x)
#Testing of Model Performances
xtab <- table(pred,y)
#Compute Confusion matrix our svm model
confusionMatrix(xtab)
```

```
## Confusion Matrix and Statistics
##
##          y
## pred      agung alvin gde rischan
##   agung     445     8   9      12
##   alvin       2   406  25      14
##   gde         4    14 444      53
```

```
##   rischan    14    22  26     311
##
## Overall Statistics
##
##                Accuracy : 0.8878
##                  95% CI : (0.8723, 0.902)
##     No Information Rate : 0.2786
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8499
##  Mcnemar's Test P-Value : 0.003024
##
## Statistics by Class:
##
##                      Class: agung Class: alvin Class: gde Class: rischan
## Sensitivity                0.9570       0.9022     0.8810          0.7974
## Specificity                0.9784       0.9698     0.9456          0.9563
## Pos Pred Value             0.9388       0.9083     0.8621          0.8338
## Neg Pred Value             0.9850       0.9677     0.9536          0.9450
## Prevalence                 0.2570       0.2488     0.2786          0.2156
## Detection Rate             0.2460       0.2244     0.2454          0.1719
## Detection Prevalence       0.2620       0.2471     0.2847          0.2062
## Balanced Accuracy          0.9677       0.9360     0.9133          0.8769
```

```
#Loading Testing Data
x1 <- subset(testing, select=-label)
y1 <- testing$label


test_pred <- predict(svm_model,x1)
#see the result
xtab <- table(test_pred,y1)
#Compute Confusion matrix
confusionMatrix(xtab)
```

```
## Confusion Matrix and Statistics
##
##          y1
## test_pred agung alvin gde rischan
##   agung      55     0   4       5
##   alvin       1    37   0       3
##   gde         2     2  42       6
##   rischan     2     3   7      33
##
## Overall Statistics
##
##                Accuracy : 0.8267
##                  95% CI : (0.7674, 0.8762)
##     No Information Rate : 0.297
```

```
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7672
##  Mcnemar's Test P-Value : 0.5401
##
## Statistics by Class:
##
##                      Class: agung Class: alvin Class: gde Class: rischan
## Sensitivity                0.9167       0.8810     0.7925          0.7021
## Specificity                0.9366       0.9750     0.9329          0.9226
## Pos Pred Value             0.8594       0.9024     0.8077          0.7333
## Neg Pred Value             0.9638       0.9689     0.9267          0.9108
## Prevalence                 0.2970       0.2079     0.2624          0.2327
## Detection Rate             0.2723       0.1832     0.2079          0.1634
## Detection Prevalence       0.3168       0.2030     0.2574          0.2228
## Balanced Accuracy          0.9266       0.9280     0.8627          0.8124
```

Based on SFFS Algorithm the best features which has good performance are "MeanX", "AbsX", "MeanY", "MinY", "MeanZ", "SdZ". So in this experiment we only use those of features. The best SVM parameters for this dataset are cost=10, gamma=0.5 (after we apply SVM tune).

Only using Time Domain Features

Set working directory and load the dataset

```
setwd("D:/Dropbox/LAB/COURSE/3/ubi/data") #set working directory

features <- read.csv("tf_tot.csv",header = TRUE) #loading all of time domain features
```

Starting SFFS Algorithm

```
####### Starting SFFS Algorithm
f.task = makeClassifTask(data = features, target = "label")
rdesc = makeResampleDesc("CV")
ctrl = makeFeatSelControlSequential(method = "sffs", maxit = NA)
res = selectFeatures("classif.rpart", f.task, rdesc, control = ctrl)

## [FeatSel] Started selecting features for learner 'classif.rpart'
## With control class: FeatSelControlSequential
## Imputation value: 1
## [FeatSel] 1: 0 bits: mmce.test.mean=0.735
## [FeatSel] 2: 1 bits: mmce.test.mean=0.644
## [FeatSel] 2: 1 bits: mmce.test.mean=0.609
```

```
## [FeatSel] 2: 1 bits: mmce.test.mean=0.647
------------------

## [FeatSel] 13: 4 bits: mmce.test.mean=0.397
## [FeatSel] 13: 4 bits: mmce.test.mean=0.327
## [FeatSel] 13: 4 bits: mmce.test.mean=0.323
## [FeatSel] 13: 4 bits: mmce.test.mean=0.336
## [FeatSel] 13: 4 bits: mmce.test.mean=0.373
## [FeatSel] Result: 6 bits : mmce.test.mean=0.285
```

```
analyzeFeatSelResult(res)
```

```
## Features         : 6
## Performance      : mmce.test.mean=0.285
## MeanX, AbsX, MeanY, MinY, MeanZ, SdZ
##
## Path to optimum:
## - Features:    0  Init   :                       Perf = 0.73495  Diff: NA
 *
## - Features:    1  Add    : MinY                  Perf = 0.48181  Diff: 0.2
5314  *
## - Features:    2  Add    : MeanX                 Perf = 0.40971  Diff: 0.0
72095  *
## - Features:    3  Add    : MeanY                 Perf = 0.37393  Diff: 0.0
35784  *
## - Features:    4  Add    : SdZ                   Perf = 0.33613  Diff: 0.0
37799  *
## - Features:    5  Add    : MeanZ                 Perf = 0.28991  Diff: 0.0
46222  *
##
## Stopped, because no improving feature was found.

####### End SFFS Algorithm
```

Start to Applying SVM Classification

```
# After we apply SFFS algorithm in time domain features, we found that only 6
 features which has good performance, there are : "MeanX", "AbsX", "MeanY", "
MinY", "MeanZ", "SdZ". So in this experiment we only use those of features.
new_f <- subset(features, select=c("MeanX", "AbsX", "MeanY", "MinY", "MeanZ",
 "SdZ","label"))

splitdf <- function(dataframe, seed=NULL) {
  if (!is.null(seed)) set.seed(seed)
  index <- 1:nrow(dataframe)
  trainindex <- sample(index, trunc(length(index)*(90/100)))
  trainset <- dataframe[trainindex, ]
  testset <- dataframe[-trainindex, ]
  list(trainset=trainset,testset=testset)
}
```

```
splits <- splitdf(new_f, seed=808)
str(splits)
```

```
## List of 2
##  $ trainset:'data.frame':    1809 obs. of  7 variables:
##   ..$ MeanX: num [1:1809] 0.388 0.677 1.975 -0.295 -0.628 ...
##   ..$ AbsX : num [1:1809] 0.1384 0.0664 0.4515 1.2965 1.0418 ...
##   ..$ MeanY: num [1:1809] -9.71 -8.12 -9.84 -10.33 -9.98 ...
##   ..$ MinY : num [1:1809] -10.15 -9.01 -12.96 -14.07 -12.38 ...
##   ..$ MeanZ: num [1:1809] 1.92 3.927 -0.356 -1.271 -1.235 ...
##   ..$ SdZ  : num [1:1809] 0.65 1.436 3.012 3.276 0.893 ...
##   ..$ label: Factor w/ 4 levels "agung","alvin",..: 4 1 2 3 4 2 1 4 2 2 ..
.
##  $ testset :'data.frame':    202 obs. of  7 variables:
##   ..$ MeanX: num [1:202] 0.5974 0.0769 -0.2685 -0.3494 1.4408 ...
##   ..$ AbsX : num [1:202] 1.587 2.64 0.178 1.86 3.158 ...
##   ..$ MeanY: num [1:202] -10.95 -9.52 -9.49 -10.71 -12.04 ...
##   ..$ MinY : num [1:202] -17.86 -14.8 -9.91 -18.79 -17.55 ...
##   ..$ MeanZ: num [1:202] 1.035 0.952 3.859 1.169 2.61 ...
##   ..$ SdZ  : num [1:202] 4.45 7.22 1.19 1.69 7.22 ...
##   ..$ label: Factor w/ 4 levels "agung","alvin",..: 1 1 1 1 1 1 1 1 1 1 ..
.
```

```
lapply(splits,nrow)
```

```
## $trainset
## [1] 1809
##
## $testset
## [1] 202
```

```
lapply(splits,head)
```

```
## $trainset
##           MeanX       AbsX      MeanY       MinY      MeanZ       SdZ
## 1830  0.3881605 0.13839924  -9.713926 -10.146953  1.9198548 0.6497238
## 390   0.6772714 0.06637231  -8.122215  -9.013465  3.9272541 1.4361473
## 730   1.9745577 0.45150434  -9.838797 -12.957370 -0.3558127 3.0116910
## 1552 -0.2953630 1.29653156 -10.325177 -14.068901 -1.2712022 3.2755954
## 1976 -0.6275057 1.04180666  -9.979438 -12.380973 -1.2346527 0.8928233
## 811  -0.1500456 0.54080810 -10.087610 -11.291659  0.7369852 0.2269612
##        label
## 1830 rischan
## 390    agung
## 730    alvin
## 1552     gde
## 1976 rischan
## 811    alvin
```

```
##
## $testset
##          MeanX      AbsX      MeanY      MinY      MeanZ      SdZ label
## 5   0.59742959 1.5867704 -10.954408 -17.85851  1.0352185 4.448004 agung
## 7   0.07686185 2.6395196  -9.519404 -14.80353  0.9521123 7.219119 agung
## 17 -0.26854840 0.1779955  -9.490250  -9.90878  3.8594697 1.186131 agung
## 30 -0.34944954 1.8598252 -10.710136 -18.79459  1.1687857 1.686225 agung
## 34  1.44083419 3.1577843 -12.042180 -17.55239  2.6103309 7.215945 agung
## 42  1.27533278 0.6075398  -9.865321 -17.50970 -0.1746398 6.212753 agung
```

```
training <- splits$trainset
testing <- splits$testset

#SVM Start here
x <- subset(training, select=-label)
y <- training$label

svm_tune <- tune(svm, train.x=x, train.y=y,
                 kernel="radial", ranges=list(cost=10^(-1:2), gamma=c(.5,1,2)
))

summary(svm_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.5
##
## - best performance: 0.2073082
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1    0.1   0.5 0.3388735 0.02761753
## 2    1.0   0.5 0.2227655 0.02448603
## 3   10.0   0.5 0.2073082 0.02211028
## 4  100.0   0.5 0.2260958 0.02174847
## 5    0.1   1.0 0.3482781 0.02892765
## 6    1.0   1.0 0.2150430 0.02357730
## 7   10.0   1.0 0.2128207 0.02386454
## 8  100.0   1.0 0.2277502 0.02383854
## 9    0.1   2.0 0.3764579 0.03457708
## 10   1.0   2.0 0.2205709 0.03168064
## 11  10.0   2.0 0.2310743 0.02622156
## 12 100.0   2.0 0.2526335 0.02727779
```

```r
#creating training model
svm_model <- svm(x,y,kernel="radial", cost=10, gamma=0.5)
summary(svm_model)
```

```
##
## Call:
## svm.default(x = x, y = y, kernel = "radial", gamma = 0.5, cost = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.5
##
## Number of Support Vectors:  1064
##
##  ( 270 227 236 331 )
##
##
## Number of Classes:  4
##
## Levels:
##  agung alvin gde rischan
```

```r
pred <- predict(svm_model,x)
system.time(pred <- predict(svm_model,x))
```

```
##    user  system elapsed
##    0.11    0.00    0.11
```

```r
#Testing of Model Performances
xtab <- table(pred,y)
#Compute Confusion matrix
confusionMatrix(xtab)
```

```
## Confusion Matrix and Statistics
##
##          y
## pred      agung alvin gde rischan
##   agung     445     8   9      12
##   alvin       2   406  25      14
##   gde         4    14 444      53
##   rischan    14    22  26     311
##
## Overall Statistics
##
##                Accuracy : 0.8878
##                  95% CI : (0.8723, 0.902)
##     No Information Rate : 0.2786
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8499
##  Mcnemar's Test P-Value : 0.003024
##
## Statistics by Class:
##
##                      Class: agung Class: alvin Class: gde Class: rischan
## Sensitivity                0.9570       0.9022     0.8810         0.7974
## Specificity                0.9784       0.9698     0.9456         0.9563
## Pos Pred Value             0.9388       0.9083     0.8621         0.8338
## Neg Pred Value             0.9850       0.9677     0.9536         0.9450
## Prevalence                 0.2570       0.2488     0.2786         0.2156
## Detection Rate             0.2460       0.2244     0.2454         0.1719
## Detection Prevalence       0.2620       0.2471     0.2847         0.2062
## Balanced Accuracy          0.9677       0.9360     0.9133         0.8769
```

```
#Loading test data
x1 <- subset(testing, select=-label)
y1 <- testing$label

#Applying SVM with testing data
test_pred <- predict(svm_model,x1)
#see the result
xtab <- table(test_pred,y1)
#Compute Confusion matrix
confusionMatrix(xtab)
```

```
## Confusion Matrix and Statistics
##
##          y1
## test_pred agung alvin gde rischan
##   agung      55     0   4       5
##   alvin       1    37   0       3
##   gde         2     2  42       6
##   rischan     2     3   7      33
##
## Overall Statistics
##
##                Accuracy : 0.8267
##                  95% CI : (0.7674, 0.8762)
##     No Information Rate : 0.297
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7672
##  Mcnemar's Test P-Value : 0.5401
##
## Statistics by Class:
##
```

```
##                      Class: agung Class: alvin Class: gde Class: rischan
## Sensitivity                0.9167       0.8810     0.7925          0.7021
## Specificity                0.9366       0.9750     0.9329          0.9226
## Pos Pred Value             0.8594       0.9024     0.8077          0.7333
## Neg Pred Value             0.9638       0.9689     0.9267          0.9108
## Prevalence                 0.2970       0.2079     0.2624          0.2327
## Detection Rate             0.2723       0.1832     0.2079          0.1634
## Detection Prevalence       0.3168       0.2030     0.2574          0.2228
## Balanced Accuracy          0.9266       0.9280     0.8627          0.8124
```

After we apply SFFS algorithm in time domain features, we found that only 6 features which has good performance, there are : "MeanX", "AbsX", "MeanY", "MinY", "MeanZ", "SdZ". So in this experiment we only use those of features.

Only Using FFT Features

Set working directory and load the dataset

```
setwd("D:/Dropbox/LAB/COURSE/3/ubi/data") #set working directory

features <- read.csv("fft_tot.csv",header = TRUE) #load FFT features
```

Running SFFS Algorithm

```
########SFFS Algorithm
f.task = makeClassifTask(data = features, target = "label")
rdesc = makeResampleDesc("CV")
ctrl = makeFeatSelControlSequential(method = "sffs", maxit = NA)
res = selectFeatures("classif.rpart", f.task, rdesc, control = ctrl)
```

```
## [FeatSel] Started selecting features for learner 'classif.rpart'
## With control class: FeatSelControlSequential
## Imputation value: 1
## [FeatSel] 1: 0 bits: mmce.test.mean=0.735
## [FeatSel] 2: 1 bits: mmce.test.mean=0.631
## [FeatSel] 2: 1 bits: mmce.test.mean=0.63
------------------------------------
## [FeatSel] 15: 5 bits: mmce.test.mean=0.456
## [FeatSel] 15: 5 bits: mmce.test.mean=0.354
## [FeatSel] 15: 5 bits: mmce.test.mean=0.358
## [FeatSel] 15: 5 bits: mmce.test.mean=0.44
## [FeatSel] 15: 5 bits: mmce.test.mean=0.471
## [FeatSel] 15: 5 bits: mmce.test.mean=0.364
## [FeatSel] Result: 7 bits : mmce.test.mean=0.333
```

```
analyzeFeatSelResult(res)
```

```
## Features         : 7
## Performance      : mmce.test.mean=0.333
## FFT1, FFT11, FFT32, FFT81, FFT106, FFT121, FFT122
##
## Path to optimum:
## - Features:    0  Init   :                      Perf = 0.73495  Diff: NA
 *
## - Features:    1  Add    : FFT81                Perf = 0.60812  Diff: 0.1
2683  *
## - Features:    2  Add    : FFT121               Perf = 0.48085  Diff: 0.1
2728  *
## - Features:    3  Add    : FFT1                 Perf = 0.42219  Diff: 0.0
5866  *
## - Features:    4  Add    : FFT32                Perf = 0.36499  Diff: 0.0
57192  *
## - Features:    5  Add    : FFT122               Perf = 0.35356  Diff: 0.0
11433  *
## - Features:    6  Add    : FFT11                Perf = 0.34063  Diff: 0.0
1293  *
##
## Stopped, because no improving feature was found.

#####End
```

Start to Applying SVM Classification

```
#
new_f <- subset(features, select=c("FFT1", "FFT13", "FFT71", "FFT81", "FFT82"
, "FFT121","label"))

splitdf <- function(dataframe, seed=NULL) {
  if (!is.null(seed)) set.seed(seed)
  index <- 1:nrow(dataframe)
  trainindex <- sample(index, trunc(length(index)*(90/100)))
  trainset <- dataframe[trainindex, ]
  testset <- dataframe[-trainindex, ]
  list(trainset=trainset,testset=testset)
}


splits <- splitdf(new_f, seed=808)
str(splits)

## List of 2
##  $ trainset:'data.frame':    1809 obs. of  7 variables:
##   ..$ FFT1  : num [1:1809] 279 -110 624 -306 -120 ...
##   ..$ FFT13 : num [1:1809] 0.526 -0.212 0.107 -2.816 -2.213 ...
##   ..$ FFT71 : num [1:1809] -0.441 -4.479 0.668 3.107 0.374 ...
##   ..$ FFT81 : num [1:1809] 629 678 1259 -128 -212 ...
##   ..$ FFT82 : num [1:1809] 46.97 70.9 3.56 737.02 214.09 ...
```

```
##   ..$ FFT121: num [1:1809] 2721 2774 3695 3088 2787 ...
##   ..$ label : Factor w/ 4 levels "agung","alvin",..: 4 1 2 3 4 2 1 4 2 2 .
..
##  $ testset :'data.frame':    202 obs. of  7 variables:
##   ..$ FFT1  : num [1:202] 162.8 30.8 -73.8 -114.1 216 ...
##   ..$ FFT13 : num [1:202] 1.3379 -9.9532 -0.1492 1.9831 -0.0984 ...
##   ..$ FFT71 : num [1:202] -0.0871 -2.9484 0.2034 3.0401 -1.9826 ...
##   ..$ FFT81 : num [1:202] 215.7 178.3 968.6 671.1 -79.7 ...
##   ..$ FFT82 : num [1:202] 318.7 929 59.5 330.1 595.8 ...
##   ..$ FFT121: num [1:202] 3160 3322 2623 3186 2689 ...
##   ..$ label : Factor w/ 4 levels "agung","alvin",..: 1 1 1 1 1 1 1 1 1 1 .
..
```

```
lapply(splits,nrow)
```

```
## $trainset
## [1] 1809
##
## $testset
## [1] 202
```

```
lapply(splits,head)
```

```
## $trainset
##           FFT1      FFT13      FFT71      FFT81     FFT82   FFT121   label
## 1830  279.3381  0.5258697 -0.4414552  629.45344  46.96617 2721.384 rischan
## 390  -109.7629 -0.2120889 -4.4789268  678.15167  70.90440 2773.542   agung
## 730   624.2164  0.1073118  0.6680689 1259.26459   3.55939 3694.770   alvin
## 1552 -306.4335 -2.8159395  3.1071807 -127.78372 737.01924 3088.004     gde
## 1976 -120.1503 -2.2130972  0.3738504 -211.81213 214.08793 2786.554 rischan
## 811   416.0572 -1.0472556 -2.2925877   35.79769 248.79900 2434.186   alvin
##
## $testset
##          FFT1       FFT13       FFT71     FFT81     FFT82   FFT121 label
## 5   162.78629  1.33794426 -0.08712095 215.68364 318.69661 3160.126 agung
## 7    30.78293 -9.95318698 -2.94837286 178.25164 928.99552 3322.033 agung
## 17  -73.75507 -0.14918545  0.20341795 968.64974  59.54426 2622.793 agung
## 30 -114.12031  1.98312984  3.04009341 671.11053 330.09111 3186.407 agung
## 34  216.04808 -0.09844912 -1.98258196 -79.72554 595.79374 2688.648 agung
## 42  249.37061 -5.61273886 -2.23313470 -11.25904 321.84320 2633.555 agung
```

```
training <- splits$trainset
testing <- splits$testset


#SVM -> Creating Model
x <- subset(training, select=-label)
y <- training$label

svm_tune <- tune(svm, train.x=x, train.y=y,
```

```
                  kernel="radial", ranges=list(cost=10^(-1:2), gamma=c(.5,1,2)
))

summary(svm_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     1
##
## - best performance: 0.2332934
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1    0.1   0.5 0.3659638 0.04124271
## 2    1.0   0.5 0.2648005 0.02849125
## 3   10.0   0.5 0.2371639 0.02495950
## 4  100.0   0.5 0.2614672 0.02175096
## 5    0.1   1.0 0.3648435 0.04637525
## 6    1.0   1.0 0.2332934 0.02330709
## 7   10.0   1.0 0.2448987 0.02330486
## 8  100.0   1.0 0.2515316 0.03977526
## 9    0.1   2.0 0.4179220 0.02355243
## 10   1.0   2.0 0.2509761 0.02442149
## 11  10.0   2.0 0.2642357 0.02742535
## 12 100.0   2.0 0.2625813 0.02600427
```

```
#creating training model
svm_model <- svm(x,y,kernel="radial", cost=1, gamma=1)
summary(svm_model)
```

```
##
## Call:
## svm.default(x = x, y = y, kernel = "radial", gamma = 1, cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##
## Number of Support Vectors:  1385
##
##  ( 319 309 322 435 )
##
```

```
##
## Number of Classes:  4
##
## Levels:
##  agung alvin gde rischan
```

```
pred <- predict(svm_model,x)
system.time(pred <- predict(svm_model,x))
```

```
##    user  system elapsed
##    0.14    0.00    0.14
```

```
#Testing of Model Performances
xtab <- table(pred,y)
#Compute Confusion matrix
confusionMatrix(xtab)
```

```
## Confusion Matrix and Statistics
##
##          y
## pred      agung alvin gde rischan
##   agung     437     9  10      23
##   alvin       9   397  28      26
##   gde         0    22 433      57
##   rischan    19    22  33     284
##
## Overall Statistics
##
##                Accuracy : 0.8574
##                  95% CI : (0.8404, 0.8732)
##     No Information Rate : 0.2786
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8091
##  Mcnemar's Test P-Value : 0.00666
##
## Statistics by Class:
##
##                      Class: agung Class: alvin Class: gde Class: rischan
## Sensitivity                0.9398       0.8822     0.8591          0.7282
## Specificity                0.9688       0.9536     0.9395          0.9479
## Pos Pred Value             0.9123       0.8630     0.8457          0.7933
## Neg Pred Value             0.9789       0.9607     0.9453          0.9269
## Prevalence                 0.2570       0.2488     0.2786          0.2156
## Detection Rate             0.2416       0.2195     0.2394          0.1570
## Detection Prevalence       0.2648       0.2543     0.2830          0.1979
## Balanced Accuracy          0.9543       0.9179     0.8993          0.8380
```

```
#SVM with real testing data
x1 <- subset(testing, select=-label)
```

```r
y1 <- testing$label


test_pred <- predict(svm_model,x1)
#see the result
xtab <- table(test_pred,y1)
#Compute Confusion matrix
confusionMatrix(xtab)
```

```
## Confusion Matrix and Statistics
##
##           y1
## test_pred agung alvin gde rischan
##   agung      48     3   6       3
##   alvin       1    31   4       4
##   gde         6     6  37      11
##   rischan     5     2   6      29
##
## Overall Statistics
##
##                Accuracy : 0.7178
##                  95% CI : (0.6504, 0.7787)
##     No Information Rate : 0.297
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6209
##  Mcnemar's Test P-Value : 0.6716
##
## Statistics by Class:
##
##                      Class: agung Class: alvin Class: gde Class: rischan
## Sensitivity                0.8000       0.7381     0.6981          0.6170
## Specificity                0.9155       0.9437     0.8456          0.9161
## Pos Pred Value             0.8000       0.7750     0.6167          0.6905
## Neg Pred Value             0.9155       0.9321     0.8873          0.8875
## Prevalence                 0.2970       0.2079     0.2624          0.2327
## Detection Rate             0.2376       0.1535     0.1832          0.1436
## Detection Prevalence       0.2970       0.1980     0.2970          0.2079
## Balanced Accuracy          0.8577       0.8409     0.7719          0.7666
```

This documentation shows that when we use all features then we apply SFFS algorithm the best chosen features are "MeanX", "AbsX", "MeanY", "MinY", "MeanZ", "SdZ", No FFT features that become best features. When we apply SFFS algorithm to time domain features and we also get the best features are "MeanX", "AbsX", "MeanY", "MinY", "MeanZ", "SdZ". Its same.

Based on that result, we can conclude that the best features for all features are : "MeanX", "AbsX", "MeanY", "MinY", "MeanZ", "SdZ". But we still wonder because when we tried to apply SFFS algorithm to frequency domain features, we get 6 best features from FFT : "FFT1", "FFT13", "FFT71", "FFT81", "FFT82", "FFT121". So, we also tried to using combine features, 6 best features and also the best FFT features.

The result can be seen below:

Start to Applying SVM Classification

```
#
new_f <- subset(features, select=c("MeanX", "AbsX", "MeanY", "MinY", "MeanZ",
 "SdZ","FFT1", "FFT13", "FFT71", "FFT81", "FFT82", "FFT121","label"))




splitdf <- function(dataframe, seed=NULL) {
  if (!is.null(seed)) set.seed(seed)
  index <- 1:nrow(dataframe)
  trainindex <- sample(index, trunc(length(index)*(90/100)))
  trainset <- dataframe[trainindex, ]
  testset <- dataframe[-trainindex, ]
  list(trainset=trainset,testset=testset)
}


splits <- splitdf(new_f, seed=808)
str(splits)
```

```
## List of 2
##  $ trainset:'data.frame':    1809 obs. of  13 variables:
##   ..$ MeanX : num [1:1809] 0.388 0.677 1.975 -0.295 -0.628 ...
##   ..$ AbsX  : num [1:1809] 0.1384 0.0664 0.4515 1.2965 1.0418 ...
##   ..$ MeanY : num [1:1809] -9.71 -8.12 -9.84 -10.33 -9.98 ...
##   ..$ MinY  : num [1:1809] -10.15 -9.01 -12.96 -14.07 -12.38 ...
##   ..$ MeanZ : num [1:1809] 1.92 3.927 -0.356 -1.271 -1.235 ...
##   ..$ SdZ   : num [1:1809] 0.65 1.436 3.012 3.276 0.893 ...
##   ..$ FFT1  : num [1:1809] 279 -110 624 -306 -120 ...
##   ..$ FFT13 : num [1:1809] 0.526 -0.212 0.107 -2.816 -2.213 ...
##   ..$ FFT71 : num [1:1809] -0.441 -4.479 0.668 3.107 0.374 ...
##   ..$ FFT81 : num [1:1809] 629 678 1259 -128 -212 ...
##   ..$ FFT82 : num [1:1809] 46.97 70.9 3.56 737.02 214.09 ...
##   ..$ FFT121: num [1:1809] 2721 2774 3695 3088 2787 ...
##   ..$ label : Factor w/ 4 levels "agung","alvin",..: 4 1 2 3 4 2 1 4 2 2 .
..
##  $ testset :'data.frame':    202 obs. of  13 variables:
##   ..$ MeanX : num [1:202] 0.5974 0.0769 -0.2685 -0.3494 1.4408 ...
##   ..$ AbsX  : num [1:202] 1.587 2.64 0.178 1.86 3.158 ...
```

```
##   ..$ MeanY : num [1:202] -10.95 -9.52 -9.49 -10.71 -12.04 ...
##   ..$ MinY  : num [1:202] -17.86 -14.8 -9.91 -18.79 -17.55 ...
##   ..$ MeanZ : num [1:202] 1.035 0.952 3.859 1.169 2.61 ...
##   ..$ SdZ   : num [1:202] 4.45 7.22 1.19 1.69 7.22 ...
##   ..$ FFT1  : num [1:202] 162.8 30.8 -73.8 -114.1 216 ...
##   ..$ FFT13 : num [1:202] 1.3379 -9.9532 -0.1492 1.9831 -0.0984 ...
##   ..$ FFT71 : num [1:202] -0.0871 -2.9484 0.2034 3.0401 -1.9826 ...
##   ..$ FFT81 : num [1:202] 215.7 178.3 968.6 671.1 -79.7 ...
##   ..$ FFT82 : num [1:202] 318.7 929 59.5 330.1 595.8 ...
##   ..$ FFT121: num [1:202] 3160 3322 2623 3186 2689 ...
##   ..$ label : Factor w/ 4 levels "agung","alvin",..: 1 1 1 1 1 1 1 1 1 1 .
..
```

```
lapply(splits,nrow)
```

```
## $trainset
## [1] 1809
##
## $testset
## [1] 202
```

```
lapply(splits,head)
```

```
## $trainset
##           MeanX       AbsX      MeanY       MinY      MeanZ       SdZ
## 1830  0.3881605 0.13839924  -9.713926 -10.146953  1.9198548 0.6497238
## 390   0.6772714 0.06637231  -8.122215  -9.013465  3.9272541 1.4361473
## 730   1.9745577 0.45150434  -9.838797 -12.957370 -0.3558127 3.0116910
## 1552 -0.2953630 1.29653156 -10.325177 -14.068901 -1.2712022 3.2755954
## 1976 -0.6275057 1.04180666  -9.979438 -12.380973 -1.2346527 0.8928233
## 811  -0.1500456 0.54080810 -10.087610 -11.291659  0.7369852 0.2269612
##           FFT1      FFT13      FFT71      FFT81     FFT82   FFT121   label
## 1830  279.3381  0.5258697 -0.4414552  629.45344  46.96617 2721.384 rischan
## 390  -109.7629 -0.2120889 -4.4789268  678.15167  70.90440 2773.542   agung
## 730   624.2164  0.1073118  0.6680689 1259.26459   3.55939 3694.770   alvin
## 1552 -306.4335 -2.8159395  3.1071807 -127.78372 737.01924 3088.004     gde
## 1976 -120.1503 -2.2130972  0.3738504 -211.81213 214.08793 2786.554 rischan
## 811   416.0572 -1.0472556 -2.2925877   35.79769 248.79900 2434.186   alvin
##
## $testset
##          MeanX      AbsX      MeanY      MinY      MeanZ      SdZ
## 5   0.59742959 1.5867704 -10.954408 -17.85851  1.0352185 4.448004
## 7   0.07686185 2.6395196  -9.519404 -14.80353  0.9521123 7.219119
## 17 -0.26854840 0.1779955  -9.490250  -9.90878  3.8594697 1.186131
## 30 -0.34944954 1.8598252 -10.710136 -18.79459  1.1687857 1.686225
## 34  1.44083419 3.1577843 -12.042180 -17.55239  2.6103309 7.215945
## 42  1.27533278 0.6075398  -9.865321 -17.50970 -0.1746398 6.212753
##          FFT1       FFT13       FFT71     FFT81     FFT82   FFT121 label
## 5   162.78629  1.33794426 -0.08712095 215.68364 318.69661 3160.126 agung
## 7    30.78293 -9.95318698 -2.94837286 178.25164 928.99552 3322.033 agung
```

```
## 17  -73.75507 -0.14918545  0.20341795 968.64974  59.54426 2622.793 agung
## 30 -114.12031  1.98312984  3.04009341 671.11053 330.09111 3186.407 agung
## 34  216.04808 -0.09844912 -1.98258196 -79.72554 595.79374 2688.648 agung
## 42  249.37061 -5.61273886 -2.23313470 -11.25904 321.84320 2633.555 agung
```

```
training <- splits$trainset
testing <- splits$testset


#SVM -> Creating Model
x <- subset(training, select=-label)
y <- training$label

svm_tune <- tune(svm, train.x=x, train.y=y,
                 kernel="radial", ranges=list(cost=10^(-1:2), gamma=c(.5,1,2)
))

summary(svm_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.2183794
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1    0.1   0.5 0.4477624 0.02720257
## 2    1.0   0.5 0.2183794 0.02725955
## 3   10.0   0.5 0.2200307 0.02622792
## 4  100.0   0.5 0.2438060 0.02601445
## 5    0.1   1.0 0.6157766 0.04051612
## 6    1.0   1.0 0.2830540 0.04353437
## 7   10.0   1.0 0.2697913 0.03616440
## 8  100.0   1.0 0.2825138 0.04156274
## 9    0.1   2.0 0.6744045 0.03174124
## 10   1.0   2.0 0.3891774 0.02966737
## 11  10.0   2.0 0.3631983 0.03616244
## 12 100.0   2.0 0.3687324 0.04417426
```

```
#creating training model
svm_model <- svm(x,y,kernel="radial", cost=10, gamma=0.5)
summary(svm_model)
```

```
##
## Call:
## svm.default(x = x, y = y, kernel = "radial", gamma = 0.5, cost = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.5
##
## Number of Support Vectors:  1409
##
##  ( 299 357 307 446 )
##
##
## Number of Classes:  4
##
## Levels:
##  agung alvin gde rischan
```

```
pred <- predict(svm_model,x)
system.time(pred <- predict(svm_model,x))
```

```
##    user  system elapsed
##    0.17    0.00    0.17
```

```
#Testing of Model Performances
xtab <- table(pred,y)
#Compute Confusion matrix
confusionMatrix(xtab)
```

```
## Confusion Matrix and Statistics
##
##          y
## pred      agung alvin gde rischan
##   agung     465     0   0       2
##   alvin       0   445   4       1
##   gde         0     4 488      22
##   rischan     0     1  12     365
##
## Overall Statistics
##
##                Accuracy : 0.9746
##                  95% CI : (0.9662, 0.9813)
##     No Information Rate : 0.2786
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.966
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: agung Class: alvin Class: gde Class: rischan
## Sensitivity                1.0000       0.9889     0.9683         0.9359
## Specificity                0.9985       0.9963     0.9801         0.9908
## Pos Pred Value             0.9957       0.9889     0.9494         0.9656
## Neg Pred Value             1.0000       0.9963     0.9876         0.9825
## Prevalence                 0.2570       0.2488     0.2786         0.2156
## Detection Rate             0.2570       0.2460     0.2698         0.2018
## Detection Prevalence       0.2582       0.2488     0.2841         0.2090
## Balanced Accuracy          0.9993       0.9926     0.9742         0.9634
```

```
#SVM with real testing data
x1 <- subset(testing, select=-label)
y1 <- testing$label


test_pred <- predict(svm_model,x1)
#see the result
xtab <- table(test_pred,y1)
#Compute Confusion matrix
confusionMatrix(xtab)
```

```
## Confusion Matrix and Statistics
##
##          y1
## test_pred agung alvin gde rischan
##   agung      49     1   1       2
##   alvin       0    32   1       1
##   gde         8     6  44       8
##   rischan     3     3   7      36
##
## Overall Statistics
##
##                Accuracy : 0.797
##                  95% CI : (0.7349, 0.8502)
##     No Information Rate : 0.297
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.7275
##  Mcnemar's Test P-Value : 0.08003
##
## Statistics by Class:
##
##                      Class: agung Class: alvin Class: gde Class: rischan
## Sensitivity                0.8167       0.7619     0.8302         0.7660
## Specificity                0.9718       0.9875     0.8523         0.9161
## Pos Pred Value             0.9245       0.9412     0.6667         0.7347
```

```
## Neg Pred Value              0.9262       0.9405     0.9338          0.9281
## Prevalence                  0.2970       0.2079     0.2624          0.2327
## Detection Rate              0.2426       0.1584     0.2178          0.1782
## Detection Prevalence        0.2624       0.1683     0.3267          0.2426
## Balanced Accuracy           0.8942       0.8747     0.8413          0.8410
```

The accuracy using real testing data is 79 %, not better than previous one.

Conclusion:

The best accuracy that we achieved is `0.8267 (82 %) using 6 best features from time domain features: "MeanX", "AbsX", "MeanY", "MinY", "MeanZ", "SdZ".`

1. `Mean of Signal X`
2. `Absolute different between gait cycle signal X`
3. `Mean of Signal Y`
4. `Minimum acceleration of Signal Y`
5. `Mean of Signal Z`
6. `Standard Deviation each gait cycle of signal Z`