Project Report

Explore Statistical Methods to Predict Heart Arrhythmia

Hari Nath Bingi

29/05/2020

Overview of the problem

Diagnosis of a heart arrhythmia involves measuring the heart activity for irregular heart beat using Electrocardiogram (ECG) and then analysing the recorded data. These parameters coupled with patient information can then be used by doctors to identify arrhythmia and its category.

Some challenges in identifying arrhythmia are:

- Many of the current algorithms are rule based implementation but the cardio log's classification is different and better.
- Impossible for a doctor to identify minute steeps and irregularities due to the high number of parameters (i.e. > 270) involved.
- **90% of clinical alarms in intensive care units might be false**. This high percentage negatively impacts both patients and clinical staff. The alarm overload might also lead to desensitization and could result in true alarms being ignored.

Hence we aim to create a classification model which will draw conclusions from the cardio log's data as a gold standard and will distinguish between the presence and absence of cardiac arrhythmia and to classify it in one of the 16 groups from ECG data of new patients. This will help in achieving the below.

- Reducing false alarms which in turn helps clinical staff to focus on the attention required areas.
- Accurate detection to expedite patient's treatment.

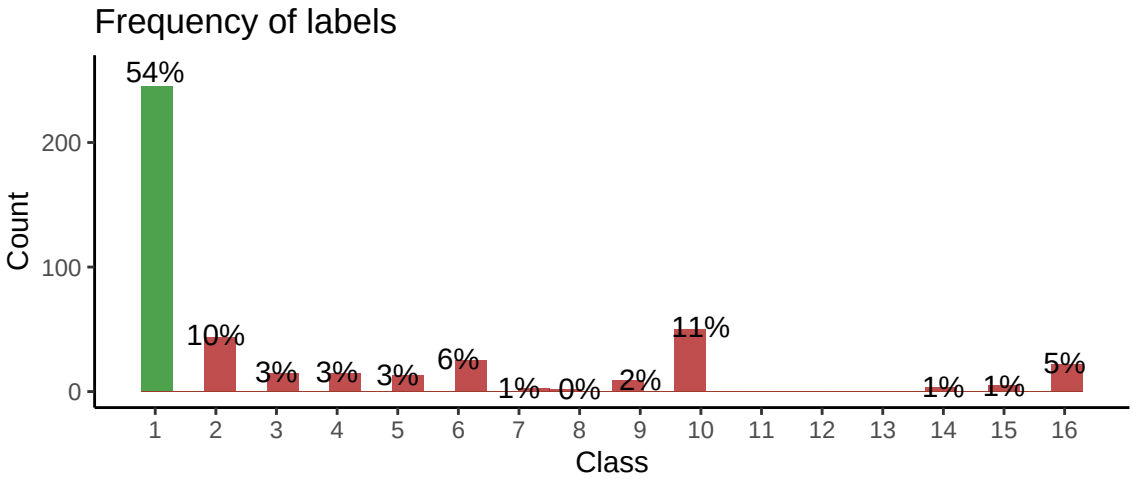
Dataset

The data has been captured as a part of study by H. Altay Guvenir and is available in uci Machine Learning repository.

```
arrhythmia_df = read.csv("arrhythmia.data",header=FALSE,na.strings=c("?"))
```

Our data has 452 observations and 279 features for each observation.

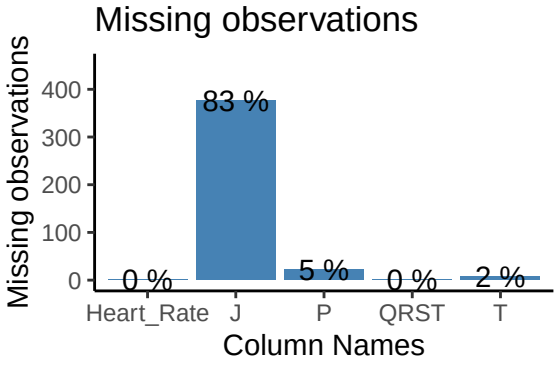
No. of Features	Type	Example Features
110	Categorical	Sex, Rag_R_Nom, Diph_R_Nom, Rag_P_Nom, Diph_P_Nom, Rag_T_Nom
169	Numeric	Age, Height, Weight, QRS_Dur



The above histogram describes the distribution of labels. Class 01 refers to 'normal' which constitute 54% of our data. The contribution for various classes of arrhythmia is less, for class 11,12,13 it is 0. The code for this is available in appendix.

Class Label	Class Description
1	Normal
2	Ischemic changes (Coronary Artery Disease)
3	Old Anterior Myocardial Infarction
4	Old Inferior Myocardial Infarction
5	Sinus tachycardy
6	Sinus bradycardy
7	Ventricular Premature Contraction (PVC)
8	Supraventricular Premature Contraction
9	Left bundle branch block
10	Right bundle branch block
11	1 degree AtrioVentricular block
12	2 degree AV block
13	3 degree AV block
14	Left ventricule hypertrophy
15	Atrial Fibrillation or Flutter
16	Others

Missing Values



From the above bar plot it is clear that we have missing values in 5 columns with column J having missing values for around 83% of observations. The code for this is available in appendix.

Challenges and Mitigation

Challenge	Mitigation
High number of features as compared to limited number of training data points	We will try dimensionality reduction usi
Missing values for numerical data columns	We will use various imputation techniqu
Heavily biased towards the normal cases	We will perform over sampling to balanc

Data Cleaning

Handling Missing Values

Column	Action Taken
Heart_Rate	There are two records for a male patient with age 75, height 190 and weight 80 and the Heart_Rate for one
J	J contributes to QRS duration (by defintion of ECG waves, please refer to below figure) which is available f
P	Seems like Missing Completely At Random, I have used knn imputation to replace missing values.
QRST	Seems like Missing Completely At Random, I have used median of other values to replace missing values.
T	Seems like Missing Completely At Random, I have used knn imputation to replace missing values.

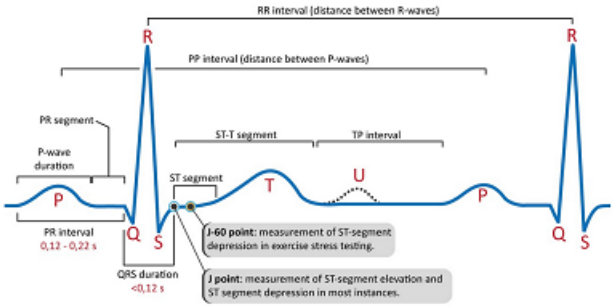


Figure 1: ECG Waveform

Table explaining missing Heart_Rate data

```
 #Two records for a male patient, age 75, height 190, weight 80 and Heart_Rate for one is missing
dplyr::filter(arrhythmia_df_nolabels[,c("Age","Sex","Height","Weight","Heart_Rate")],(Age==75 & Sex == 
```

```
##   Age Sex Height Weight Heart_Rate
## 1  75   0    190     80         63
## 2  75   0    190     80         NA
```

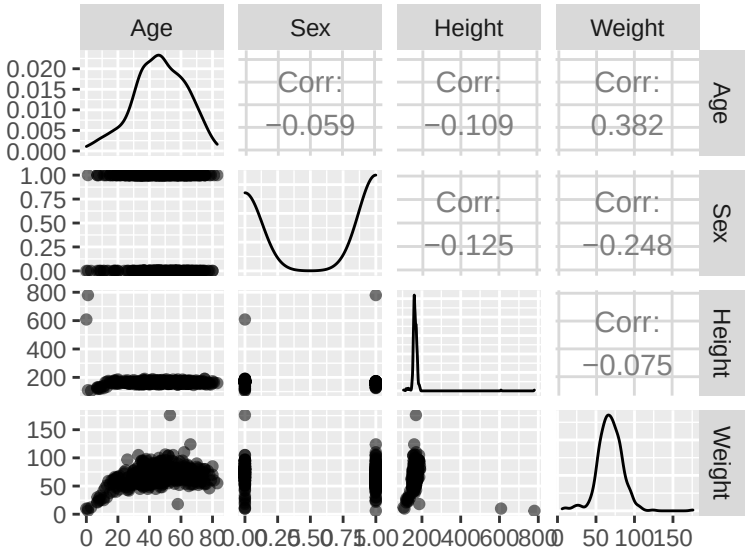
Data cleanup actions are perfomed below.

```
processed_arrhythmia_df_nolabels = arrhythmia_df_nolabels[,-which(names(arrhythmia_df_nolabels) %in% c(
processed_arrhythmia_df_nolabels$Heart_Rate[is.na(processed_arrhythmia_df_nolabels$Heart_Rate)] = 63 #V
processed_arrhythmia_df_nolabels$QRST = Hmisc::impute(processed_arrhythmia_df_nolabels$QRST, median)  #
library(VIM)
```

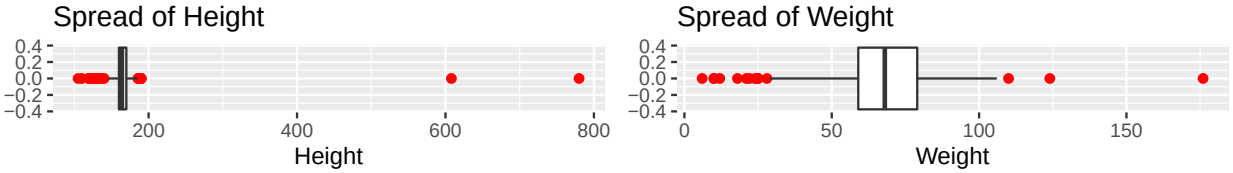
```
processed_arrhythmia_df_nolabels = kNN(processed_arrhythmia_df_nolabels,
                                       variable = c("P", "T"),
                                       k = 6) #knn impute column P,T
processed_arrhythmia_df_nolabels = processed_arrhythmia_df_nolabels[,-which(names(processed_arrhythmia_
```

Outlier Examination

```
library(GGally)
ggpairs(arrhythmia_df_nolabels[,c('Age', 'Sex', 'Height', 'Weight')], aes(alpha = 0.8))
```

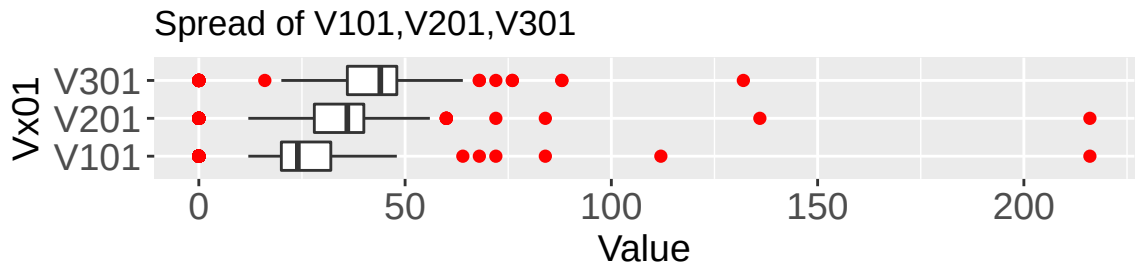


Examining the pair wise plots for height and weight i.e. the 2 left bottom ones indicate there are some outliers in Height and Weight attributes and hence will examine them via boxplots. The code for box plot is in the appendix.



Column	Outliers	Action Plan
Height	2 observation more than 600	does not seem realistic for height more than 600 centimetres and hence I have su
Weight	1 observation around 160	160 kilograms seems to be realistic and hence I have taken no action.

```
processed_arrhythmia_df_nolabels$Height[which(processed_arrhythmia_df_nolabels$Height > 600)] = median(
```

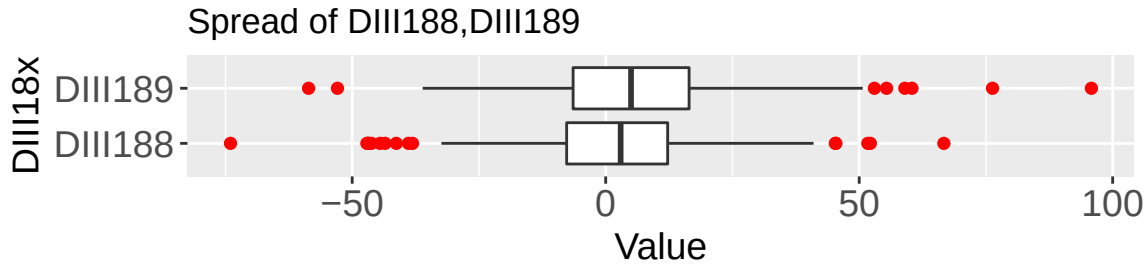


V101, V201,V301 indicates some outliers and there seems to be a pattern i.e. one highest value for all. The code for this is available in appendix. Hence examining the highest value of V101.

```
arrhythmia_df[arrhythmia_df$V101 > 200,c("V101","V201","V301","class")]
```

```
##     V101 V201 V301 class
## 298  216  216  132    10
```

It looks like outlier of V101, V201, V301 belong to same data point and to a rare class of arrhythmia i.e. class 10. As they belong a rare class and hence they do not seem to fit with others but these are not outliers.



DIII188, DIII189 indicates some outliers and there seems to be a pattern i.e. one highest value for all. Hence examining the highest value of DIII188.

```
arrhythmia_df[arrhythmia_df$DIII188 > 60,c("DIII188","DIII189","class")]
```

```
##   DIII188 DIII189 class
## 6    66.7    95.8    14
```

It looks like outlier of DIII188, DIII189 belong to same data point and to a rare class of arrhythmia i.e. class 14. As they belong a rare class and hence they do not seem to fit with others but these are not outliers.

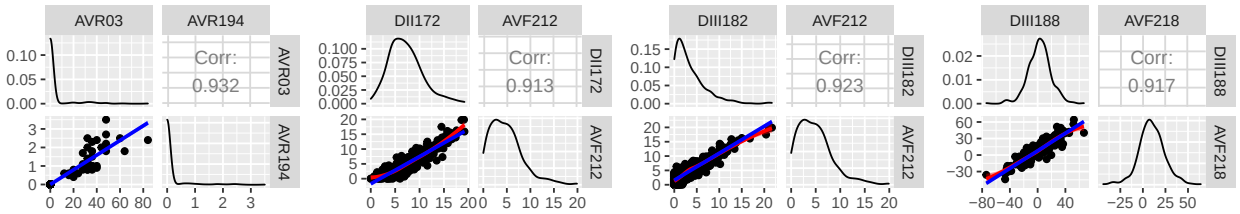
Similar features have been examined in the same manner i.e. by using side by side box plots. The code for the same is available in appendix.

Identify correlated features

```
cormatrix_res <- cor(processed_arrhythmia_df_nolabels[, nonzero_var]) #Creating a correlation matrix fo
x1 = y1 = c()
for (i in seq(nrow(cormatrix_res)))
{
```

```
  for (j in seq(ncol(cormatrix_res)))
  {
    if (cormatrix_res[i, j] > 0.9 & i != j) #Identifying features which have a  correlation of more tha
    {
      x1 = c(x1, rownames(cormatrix_res)[i])  #Rowname, colname of the matrix include the feature names
      y1 = c(y1, colnames(cormatrix_res)[j])
    }
  }
}
corelationcols = data.frame(x1, y1) #Creating a data frame
related_numerical_cols = corelationcols[!duplicated(t(apply(corelationcols, 1, sort))), ] #Removing dup
related_numerical_cols$x1 = as.character(related_numerical_cols$x1)
related_numerical_cols$y1 = as.character(related_numerical_cols$y1)
colums_to_drop = related_numerical_cols$y1 #Saving the correlated features for future use
```

Identifying continuous ECG correlated features: My analysis revealed that the below continuous variables are correlated and hence visualizing. The code for this visualization is available in appendix



The bottom left of each graph shows the points of the respective columns and a smotting line to indicate they are related. We also have correlation described in all plot as > 0.9. Hence we can drop these columns i.e. AVR194, AVF212, AVF218.

```
library("GoodmanKruskal")
subset_arrhythmia_df <-   subset(processed_arrhythmia_df_nolabels, select = categorical_columns)
GKmatrix1 <- GKtauDataframe(subset_arrhythmia_df)  #Calculating Goodman and Kruskal's Tau measure for c
x2 = y2 = c()
for (i in seq(nrow(GKmatrix1)))
{
  for (j in seq(ncol(GKmatrix1)))
  {
    if (!is.na(GKmatrix1[i, j]))
    {
      if (GKmatrix1[i, j] > 0.9 & i != j) #Identifying features which have a forward association of mor
      {
        x2 = c(x2, rownames(GKmatrix1)[i]) #Rowname, colname of the matrix include the feature names
        y2 = c(y2, colnames(GKmatrix1)[j])
      }
    }
  }
}
corelationcols2 = data.frame(x2, y2) #Storing the values in a data frame
related_cat_cols = corelationcols2[!duplicated(t(apply(corelationcols2, 1, sort))), ] #Removing duplica
```

Identifying categorical ECG correlated parameters using Goodman and Kruskal's Tau measure: After analysis we know that the below continuous features are correlated. We have also Tau values of forward association for reference.

Column X	R'_Wave	DII04	DII04	AVR04	AVF04	AVF04	V303	V303	V303	V304	V404	V603	DII174	DII175	DIII185	AVF215	V1225	V2235	V3244	V3244	V3244
Column Y	R_Prime_Wave	AVL11	DII175	AVR195	AVL11	AVF215	V304	V3244	V3245	V3245	V4255	V6274	AVL11	AVL11	DIII04	AVL11	V104	V204	AVL08	V304	V3245
Tau(x,y)	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.916	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

```
## [1] "Hence these columns can be dropped - R_Prime_Wave, AVL11, DII175, AVR195, AVF215, V304, V3244, V
```

Evaluation metric

We will use macro averaged f1-score and macro averaged recall as metric to evaluate our model

We will calculate macro averaged f1 as a simple arithmetic mean of our per-class F1-scores.

As the data is highly skewed towards class 01 and we want our classification models to classify all labels with equal importance we are choosing macro averaged f1. We will also use macro averaged recall as a guiding factor to ensure our model is able to classify high number of relevant classes.

We have defined a function in the below code to calculate macro avergaed f1, macro averaged recall.

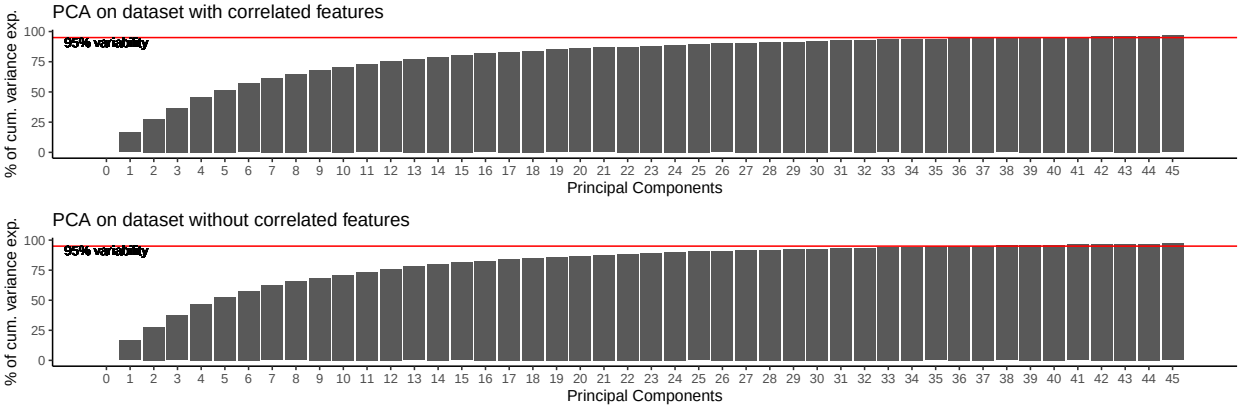
```
calcF1Scores = function(actual, predicted) {
  actual = as.numeric(actual)
  predicted = as.numeric(predicted)
  df = data.frame(actual = actual, predicted = predicted) #Creare a dataframe for actual and predicted
  fone = recall = c()
  for (i in seq(min(actual), max(actual))) {  #Calculate fone and recall for every class
    tp = nrow(df[df$predicted == i & df$actual == i, ])
    fp = nrow(df[df$predicted == i & df$actual != i, ])
    fn = nrow(df[df$predicted != i & df$actual == i, ])
    #Calculate precision recall and f1
    PR = tp / (tp + fp)
    RE = tp / (tp + fn)
    f1 = (2 * PR * RE) / (PR + RE)
    #Handle some exception scenarios
    if (tp == fp & fp == fn & fn == 0)
    {
      PR = 1
      RE = 1
      f1 = 1
    }
    else if (tp == fp & fp == 0)
    {
      PR = 1
      RE = tp / (tp + fn)
      f1 = (2 * PR * RE) / (PR + RE)

    } else if (tp == 0)
    {
      PR = 0
      RE = 0
      f1 = 0
    }
    fone = c(fone, f1)
    recall = c(recall, RE)
```

```
  }
  return(list(mean(fone), mean(recall))) #Avergae values of all classes to calculate macro f1
}
```

Dimension Reduction

Principal Component Analysis

I have performed PCA on the dataset keeping correlated columns and on the dataset without keeping correlated columns. The code for PCA is in the appendix.



We will have to use at least 39 principal components for dataset with correlated features and 36 principal compoenets for datatset without correlated features as they explain 95% variability. It seems that dropping the correlated features did not help with principal component analysis. This principal component analysis is performed on the entire data and is only for our evaluation. We will perform principal component analysis only on training data set while building classifiers. The code of cumulative variance plot is in the appendix.

Feature subset

I have performed stepwise forward selection as below:

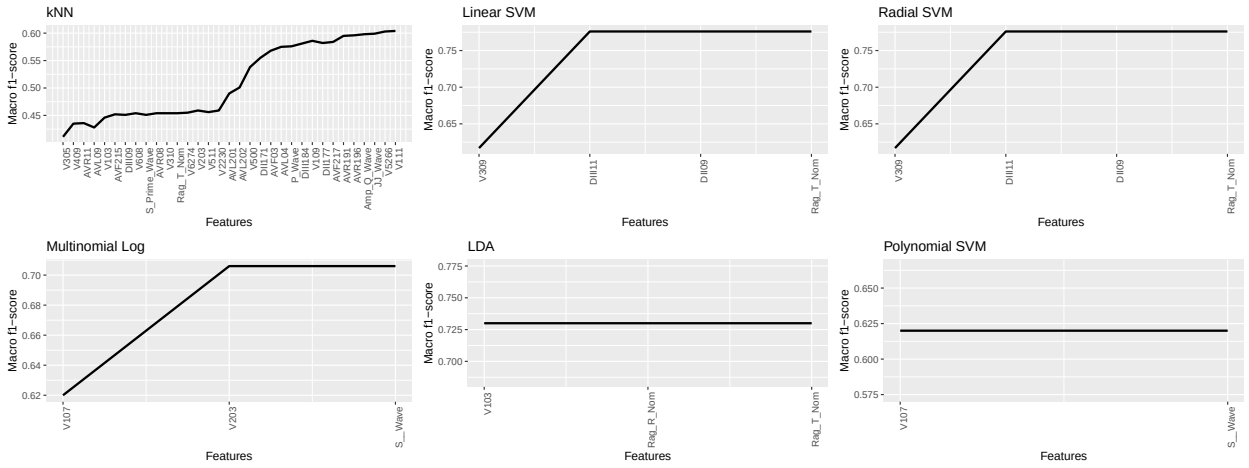
1. Split the dataset into 60% for training and 40% for validation.
2. Oversample the training data to balance classes.
3. Perform classification for each feature and calculate macro weighted f1-score on the validation data.
4. Keep the feature with best macro weighted f1-score.
5. Repeat step 3 & 4 for 40 times by keeping the best predictors and augmenting predictors for each iteration.
6. Choose minimum number of predictors with the best macro weighted f1-score.
7. Repeat step 3,4,5 & 6 for kNN, LDA, Multinomial Log, Linear SVM, Polynomial SVM, Radial SVM.

The below is the code for knn feature subset, the code for other classifier subset is in the appendix.

```
library(ROSE) #ROSE package for oversampling
selectFeature <-   #This function will return the feature with best macro f1-score
  function(train, test, cls.train, cls.test, features) {
    ## identify a feature to be selected
    current.best.fone <- -Inf
    selected.i <- NULL
```

```
    for (i in 1:ncol(train)) {
      current.f <- colnames(train)[i]
      if (!current.f %in% features) { #Verify if the feature is already in considered in the list
        #Create dataframe for classifiers to be used
        subfeatures_df_train = data.frame(class = cls.train)
        subfeatures_df_test = data.frame(class = cls.test)
        allcols = c(features, current.f)
        for (eachcol in allcols)
        {
          subfeatures_df_train[, eachcol] = train[, eachcol]
          subfeatures_df_test[, eachcol] = test[, eachcol]
        }

        #Oversample the training data to balance minor classes
        oversampled_df = subfeatures_df_train
        all_exis_classes  = sort(unique(arrhythmia_labels))
        for (each_minor_class in all_exis_classes[!all_exis_classes %in% 1]) #ovun.sample only works on
        {
          testdf = subset(oversampled_df, class == each_minor_class | class == 1)
          balanceddata <- ovun.sample(class ~ .,data = get("testdf", sys.frame(1)),method = "over",p =
          oversampled_df = rbind(  #only adding the oversampled data of minor class and not considering
            subset(balanceddata, class != 1),
            subset(oversampled_df, class != each_minor_class)
          )}

        flag <- TRUE
        tryCatch({ #sometimes knn throws "too many ties" for some variables and hence catching to proce
          preds = class::knn(oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop = F
        }, error = function(e) {
          flag <<- FALSE
        })

        if (!flag) { next} #Iterate over next feature in case knn throws an error like "too many ties"

        f1score <- calcF1Scores(as.numeric(levels(preds))[preds], cls.test) #Calculate macro f1-score
        test.fone = round(f1score[[1]], 3)
        if (test.fone > current.best.fone) { #Save the feature with best macro f1
          current.best.fone <- test.fone
          selected.i <- colnames(train)[i]
        }
      }
    }
    return(list(selected.i, current.best.fone))  #Return feature and fone score for plotting
  }

library(caret)
set.seed(1)
inTrain <- createDataPartition(arrhythmia_labels, p = .6)[[1]] #Split data into training and validation
allFeatures <- colnames(processed_arrhythmia_df_nolabels)

train <- processed_arrhythmia_df_nolabels[inTrain, ]
test  <- processed_arrhythmia_df_nolabels[-inTrain, ]
cls.train <- arrhythmia_labels[inTrain]
```

```
cls.test <- arrhythmia_labels[-inTrain]
features <- NULL
fone_scores = NULL

#Iterating 100 times to find best 100 features
for (j in 1:100) {
  featureoutput <- selectFeature(train, test, cls.train, cls.test, features)
  features <- c(features, featureoutput[[1]])
  fone_scores = c(fone_scores, featureoutput[[2]])
}
knn_featuresubset_metrics = data.frame(Features=features,macrof1=fone_scores,index=seq(1:length(features
```



Each subplot indicates the macro f1-score for an additional predictor for the respective classifier. Example - the right top plot indicates that the macro f1-score for Radial SVM is 0.58 for V301 predictor and 0.77 for V309 and DIII11 predictors. These subplots does not show the entire analysis performed but only represent the required information. The code for these subplots is available in appendix.

We could then conclude the below as best features for the respective classifiers.

Classifier	Best Features
kNN	V305, V409, AVR11, AVL09, V103, AVF215, DIII09, V608, S_Prime_Wave, AVR08, V310, Rag_T_N
LDA	V103
Multinomial Log	V107, V203
Linear SVM	V309, DIII11
Polynomial SVM	V107
Radial SVM	V309, DIII11

Approach for classification

We will try kNN, LDA, Multinomial Log, Linear SVM, Polynomial SVM, Radial SVM in 3 different approaches via cross validation.

Approach 1

1. Split the dataset with correlated features into 10 folds.
2. For every fold, consider one fold as a validation data set and the remaining as training data set.

3. Perform PCA on the training data set.
4. Identify minimum number of principal components i.e. 'p' to explain 95% variability, in most cases it is 39.
5. Oversample 'p' principal components to balance classes using ROSE package.
6. Perform kNN, LDA, Multinomial Log, Linear SVM, Polynomial SVM, Radial SVM on the 'p' principal components of training data.
7. Apply training PCA on the validation data set.
8. Select 'p' principal components of the validation data set.
9. Calculate macro weight f1 for every classifier on the validation data set principal components.
10. Repeat from step 2 for all the folds.
11. Average macro weighted f1-score of all folds to achieve one macro weighted f1 for each classifier.

The below is the code for approach 1.

```
set.seed(1)
fold <- createFolds(arrhythmia_labels, k = 10) #Create 10-fold cross-validation

#Variables to hold f1 score and recall for every hold
tmpf1knn = tmpf1linsvm = tmpf1ploy = tmpf1radc = tmpf1lda = tmpf1log = c()
tmprecaknn = tmprecalinsvm = tmprecaploy = tmprecaradc = tmprecalda = tmprecalog = c()

for (i in 1:length(fold)) {

  arrhythmia.pca <-prcomp(processed_arrhythmia_df_nolabels[-fold[[i]], ]) #Apply PCA on training set

  #Calculate cum variance
  tot.var <- sum(arrhythmia.pca$sdev ^ 2)
  var.explained <- data.frame(pc = seq(1:ncol(processed_arrhythmia_df_nolabels[-fold[[i]], ])),
                              var.explained  = arrhythmia.pca$sdev ^ 2 / tot.var)
  cum_var.explained = c()
  for (u in seq(nrow(var.explained)))
  {
    cum_var.explained =  c(cum_var.explained, sum(var.explained[1:u, 2]) * 100)
  }

  pctouse = min(which(cum_var.explained > 95)) #identify number of PCs that explain 95% variability
  #Oversample minor classes of training Principal components
  oversampled_df = data.frame(arrhythmia.pca$x[, 1:pctouse])
  oversampled_df$class = arrhythmia_labels[-fold[[i]]]
  all_exis_classes  = sort(unique(arrhythmia_labels))
  for (eachminroclass in all_exis_classes[!all_exis_classes %in% 1])
  {
    balanceddata <- ovun.sample(class ~ .,data = subset(oversampled_df, class == eachminroclass | class
    oversampled_df = rbind(subset(balanceddata, class != 1),
                           subset(oversampled_df, class != eachminroclass))
  }
  oversampled_df <- oversampled_df[sample(nrow(oversampled_df)), ]  #Shuffling rows

#apply PCA on validation dataset
  arrhythmia.pcatest = predict(arrhythmia.pca, processed_arrhythmia_df_nolabels[fold[[i]], ])

 #Training on different classfiers and calculate validation f1-score
  preds <- knn(oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop =FALSE],data.fra
  f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]]) #Calculating
```

```
  tmpf1knn = c(tmpf1knn, f1score[[1]])
  tmprecaknn = c(tmprecaknn, f1score[[2]])

  linsvm.model1 <- svm(x = oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop =FALS
  preds <- predict(linsvm.model1, newdata = data.frame(arrhythmia.pcatest[, 1:pctouse]))
  f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
  tmpf1linsvm = c(tmpf1linsvm, f1score[[1]])
  tmprecalinsvm = c(tmprecalinsvm, f1score[[2]])

  polysvm.model1 <- svm(x = oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop =FA
  preds <- predict(polysvm.model1, newdata = data.frame(arrhythmia.pcatest[, 1:pctouse]))
  f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
  tmpf1ploy = c(tmpf1ploy, f1score[[1]])
  tmprecaploy = c(tmprecaploy, f1score[[2]])

  radsvm.model1 <- svm(x = oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop =FALS
  preds <- predict(radsvm.model1, newdata = data.frame(arrhythmia.pcatest[, 1:pctouse]))
  f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
  tmpf1radc = c(tmpf1radc, f1score[[1]])
  tmprecaradc = c(tmprecaradc, f1score[[2]])

  lda.model <- MASS::lda(class ~ ., data = oversampled_df)
  preds = predict(lda.model, newdata = data.frame(arrhythmia.pcatest[, 1:pctouse]))$class
  f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
  tmpf1lda = c(tmpf1lda, f1score[[1]])
  tmprecalda = c(tmprecalda, f1score[[2]])

  multinomlogmodel <- multinom(class ~ ., data = oversampled_df, trace = FALSE)
  preds = predict(multinomlogmodel, newdata = data.frame(arrhythmia.pcatest[, 1:pctouse]))
  f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
  tmpf1log = c(tmpf1log, f1score[[1]])
  tmprecalog = c(tmprecalog, f1score[[2]])
}

fone.val = c(mean(tmpf1log),mean(tmpf1lda),mean(tmpf1knn),mean(tmpf1linsvm),mean(tmpf1ploy),mean(tmpf1ra
recall.val = c(mean(tmprecalog),mean(tmprecalda),mean(tmprecaknn),mean(tmprecalinsvm),mean(tmprecaploy)

#Store values for line plot
plot_df95pca = data.frame(index = seq(1:6),fone = fone.val,recall = recall.val,algo = c("Multinomial Log
#Store values for box plot
boxplot_df_pca95 = data.frame("Values" = c(tmprecalog,tmprecalda,tmprecaknn,tmprecalinsvm,tmprecaploy,tm
```

Approach 2

This will be same as approach 1, but with dataset excluding correlated features. The code for this is in appendix.

Approach 3

1. Split the dataset with correlated features into 10 folds.
2. For every fold, consider one fold as a validation data set and the remaining as training data set.
3. Oversample training data set to balance classes.

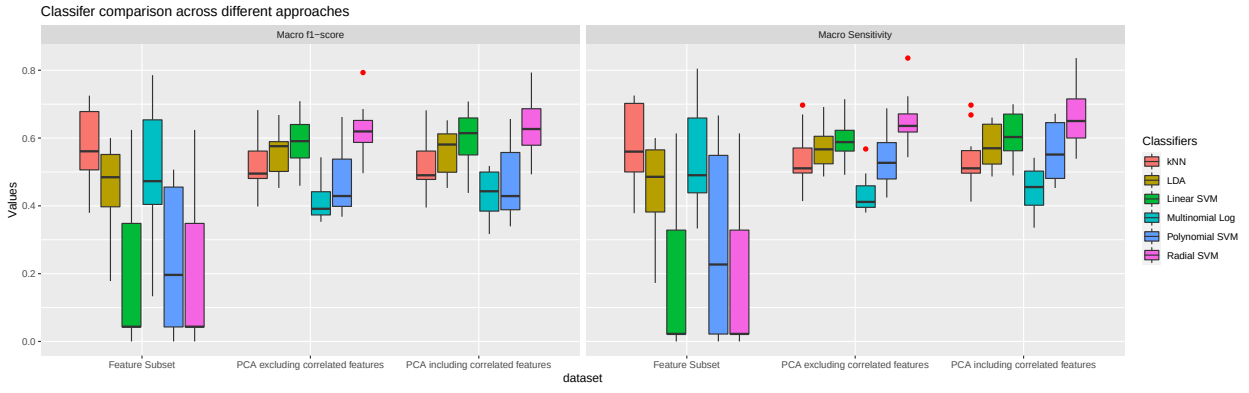
4. Select the best features (as described in above section) subset of oversampled training data respectively for every classifier.
5. Perform kNN, LDA, Multinomial Log, Linear SVM, Polynomial SVM, Radial SVM.
6. Calculate macro weight f1 for every classifier on the best features subset of validation data sets for every classifier respectively.
7. Repeat from step 2 for all the folds.
8. Average macro weighted f1-score of all folds to achieve one macro weighted f1 for each classifier.

The below is the code for the same.

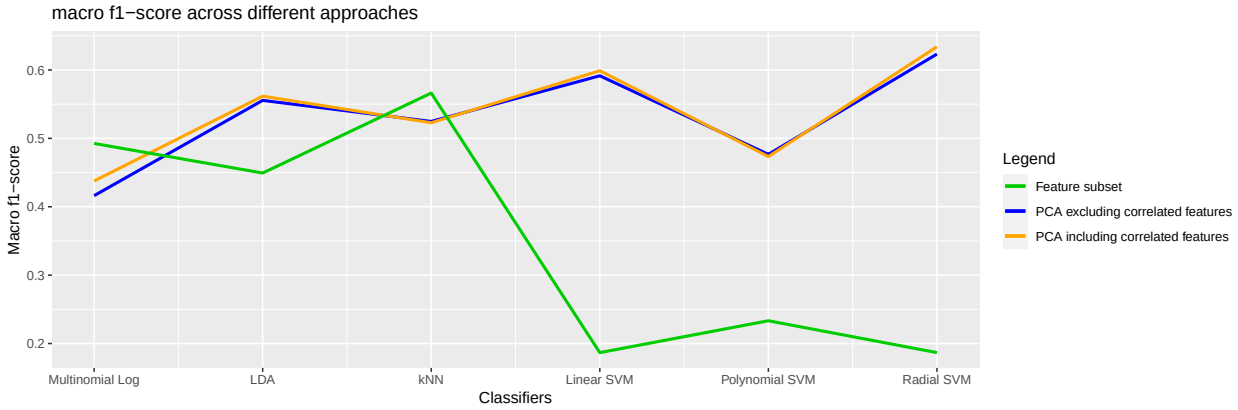
```
fold <- createFolds(arrhythmia_labels, k = 10)
# apply 10-fold cross-validation
tmp4f1knn = tmp4f1linsvm = tmp4f1ploy = tmp4f1radc = tmp4f1lda = tmp4f1log = c()
tmp4recaknn = tmp4recalinsvm = tmp4recaploy = tmp4recaradc = tmp4recalda = tmp4recalog = c()

for (i in 1:length(fold)) {
  # Oversample training data
  oversampled_df = processed_arrhythmia_df_nolabels[-fold[[i]], ]
  oversampled_df$class = arrhythmia_labels[-fold[[i]]]
  all_exis_classes  = sort(unique(arrhythmia_labels))
  for (eachminroclass in all_exis_classes[!all_exis_classes %in% 1]) #looping as ovun.sample only suppo
  {
  balanceddata <- ovun.sample(class ~ .,data = subset(oversampled_df, class == eachminroclass | class ==
  oversampled_df = rbind(subset(balanceddata, class != 1),
                         subset(oversampled_df, class != eachminroclass))
  }
 oversampled_df <- oversampled_df[sample(nrow(oversampled_df)), ]  #Shuffling rows

 knnfeatures = c("V305","V409","AVR11","AVL09","V103","AVF215","DIII09","V608","S_Prime_Wave","AVR08","
 linsvmfeatures = c("V309", "DIII11")
 polysvmfeatures =   c("V107")
 radsvmfeatures = c("V309", "DIII11")
  ldafeatures = c("V103")
  multilogfeatures = c("V107", "V203")

  preds <- knn(oversampled_df[, knnfeatures, drop = FALSE], processed_arrhythmia_df_nolabels[fold[[i]],
  f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
  tmp4f1knn = c(tmp4f1knn, f1score[[1]])
  tmp4recaknn = c(tmp4recaknn, f1score[[2]])

  linsvm.model1 <- svm(x = oversampled_df[, linsvmfeatures, drop = FALSE],y = oversampled_df$class,kern
  preds <- predict(linsvm.model1, newdata = processed_arrhythmia_df_nolabels[fold[[i]], linsvmfeatures]]
  f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
  tmp4f1linsvm = c(tmp4f1linsvm, f1score[[1]])
  tmp4recalinsvm = c(tmp4recalinsvm, f1score[[2]])

  polysvm.model1 <- svm(x = oversampled_df[, polysvmfeatures, drop = FALSE],y = oversampled_df$class,ke
  preds <- predict(polysvm.model1, newdata = processed_arrhythmia_df_nolabels[fold[[i]], polysvmfeature
  f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
  tmp4f1ploy = c(tmp4f1ploy, f1score[[1]])
  tmp4recaploy = c(tmp4recaploy, f1score[[2]])

  radsvm.model1 <- svm(x = oversampled_df[, radsvmfeatures, drop = FALSE],y = oversampled_df$class,kern
  preds <- predict(radsvm.model1, newdata = processed_arrhythmia_df_nolabels[fold[[i]], radsvmfeatures]]
```

```
  f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
  tmp4f1radc = c(tmp4f1radc, f1score[[1]])
  tmp4recaradc = c(tmp4recaradc, f1score[[2]])

  ldafeatures_withclass = c("V103", "class")
  lda.model <- MASS::lda(class ~ ., data = oversampled_df[, ldafeatures_withclass, drop = FALSE])
  preds = predict(lda.model, newdata = processed_arrhythmia_df_nolabels[fold[[i]], ldafeatures, drop = I
  f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
  tmp4f1lda = c(tmp4f1lda, f1score[[1]])
  tmp4recalda = c(tmp4recalda, f1score[[2]])

  multilogfeatures_with_class = c("V107", "V203", "class")
  multinomlogmodel <- multinom(class ~ ., data = oversampled_df[, multilogfeatures_with_class, drop =FA
  preds = predict(multinomlogmodel, newdata = processed_arrhythmia_df_nolabels[fold[[i]], multilogfeatu
  f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
  tmp4f1log = c(tmp4f1log, f1score[[1]])
  tmp4recalog = c(tmp4recalog, f1score[[2]])

}

fone.val = c(mean(tmp4f1log ),mean(tmp4f1lda),mean(tmp4f1knn),mean(tmp4f1linsvm),mean(tmp4f1ploy),mean(t
recall.val = c(mean(tmp4recalog ),mean(tmp4recalda),mean(tmp4recaknn),mean(tmp4recalinsvm),mean(tmp4reca

plot_dfsubset = data.frame(index= seq(1:6),fone=fone.val,recall=recall.val,algo= c("Multinomial Log","LI

boxplot_df_subset= data.frame("Values"=c(tmp4recalog,tmp4recalda,tmp4recaknn,tmp4recalinsvm,tmp4recaploy
```

Classification performance evaluation



This boxplot shows the variability of performance i.e. macro weighted f1 score (left) and macro sensitivity (right) across various folds for different classifiers for the 3 approaches. In general feature subset does not seem like a good option for this dataset due it it's huge variability across the folds. Performing PCA is giving better performance and less variability across folds for all classifiers. PCA performed after removing correlated features seems to be having less variability across various folds of data. kNN and multinomial log seems to be doing well with feature subsetting. Let us examine average of macro f1-scores. The code for this is in the appendix.



The above plot shows average of macro f1-score across all folds for various approached and classifiers. The average of macro f1-score over PCA including and excluding correlated features are almost similar but as per the previous plot the variability is less over PCA excluding correlated features. Hence we will not consider classification performed on PCA including correlated features. kNN and multinomial log have better performance when performed on subset of features than performed over principal components but the variation across various folds is large and also they are not better than Linear SVM and Radial SVM performed over principal components and hence we will also not consider results of feature subset. Linear SVM, Radial SVM classifiers performed on principal components excluding correlated features seems to be performing better with a macro f1-score of 0.59, 0.62. We will now optimize hyper parameters i.e. cost for linear SVM, cost and gamma for radial SVM.

Hyperparameter optimization

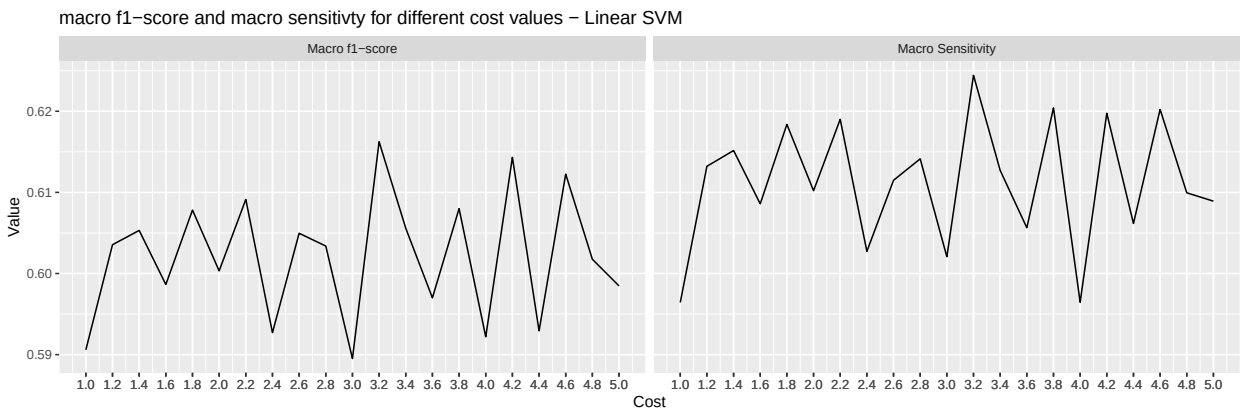
We will perform steps for optimizing hyper parameters.

1. Split the dataset without correlated features into 10 folds.
2. For every fold, consider one fold as a validation data set and the remaining as training data set.
3. Perform PCA on the training data set.
4. Identify minimum number of principal components i.e. 'p' to explain 95% variability, in most cases it is 39.
5. Oversample 'p' principal components to balance classes.
6. Perform Linear SVM with a cost of 1 on the 'p' principal components of training data.
7. Apply training PCA on the validation data set.
8. Select 'p' principal components of the validation data set.
9. Calculate macro f1-score on the validation data set principal components.
10. Repeat from step 2 for all the folds.
11. Average macro weighted f1-score of all folds to achieve one macro weighted f1 for each cost value.
12. Repeat from step 6 to step 11 for various values of cost i.e. 1,1.5,2 until 10.
13. Identify the best cost value for which macro f1-score is high.

The below is the code for linear SVM hyper parameter optimization.

```
fold <- createFolds(arrhythmia_labels, k=10) # apply 10-fold cross-validation
costval = seq(1,10,0.2) #Values of cost to be tested
for(cost in costval)
{
f1scores = c()
sens_Scores = c()
for(i in 1:length(fold)){
```

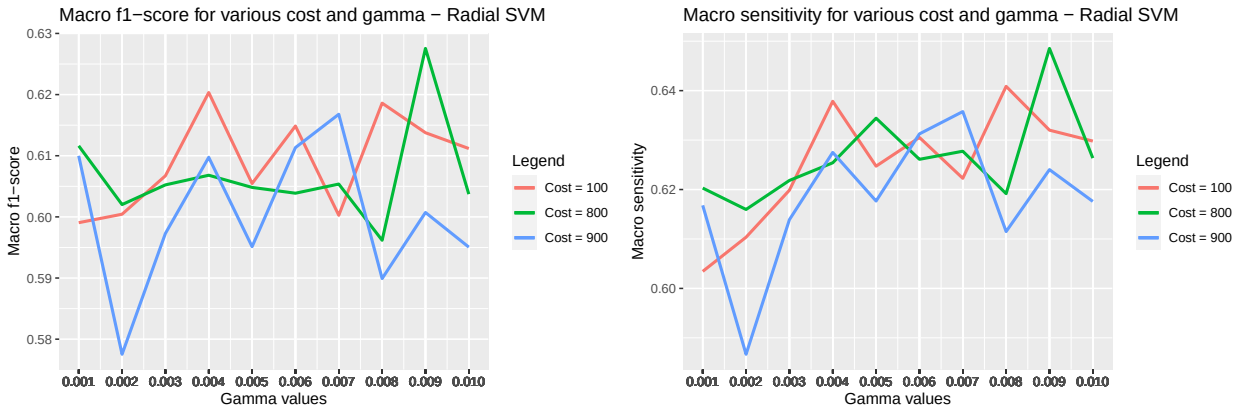
```
#Apply PCA on training dat
arrhythmia.pca <- prcomp(processed_arrhythmia_df_nolabels[-fold[[i]],-which(names(processed_arrhythmia_
tot.var <- sum(arrhythmia.pca$sdev^2)
var.explained <- data.frame(pc = seq(1:ncol(processed_arrhythmia_df_nolabels[-fold[[i]],-which(names(pr

cum_var.explained =c()
for (u in seq(nrow(var.explained)))
{cum_var.explained =  c(cum_var.explained,sum(var.explained[1:u,2])*100)}

#Identify number of PCs to be used for 95% variability
pctouse = min(which(cum_var.explained > 95))

#Oversample minor classes
oversampled_df = data.frame(arrhythmia.pca$x[,1:pctouse])
oversampled_df$class = arrhythmia_labels[-fold[[i]]]
all_exis_classes  = sort(unique(arrhythmia_labels))
for (eachminroclass in all_exis_classes[!all_exis_classes %in% 1])
{
balanceddata<-ovun.sample(class ~ ., data =subset(oversampled_df,class == eachminroclass | class == 1),
oversampled_df = rbind(subset(balanceddata,class != 1),subset(oversampled_df,class != eachminroclass))
}
oversampled_df <- oversampled_df[sample(nrow(oversampled_df)),]  #Shuffling rows

#apply PCA on test data
arrhythmia.pcatest = predict(arrhythmia.pca,processed_arrhythmia_df_nolabels[fold[[i]],-which(names(pro

svm.model1 <- svm(x = oversampled_df[,-which(colnames(oversampled_df) %in% c("class")),drop=FALSE], y =
preds <- predict(svm.model1, newdata=data.frame(arrhythmia.pcatest[,1:pctouse]) )

f1score <- calcF1Scores(as.numeric(levels(preds))[preds],arrhythmia_labels[fold[[i]]]) #Calculate f1-sc
f1scores = c(f1scores,f1score[[1]])
}
knn.fone = c(knn.fone,mean(f1scores))
knn.sen = c(knn.sen,mean(sens_Scores))
}

#Store data for plotting
lin.plot.dat <- data.frame(values = c(unlist(knn.fone),unlist(knn.sen)),metric = rep(c("Macro f1-score"
```



These plots indicate macro f1-score (left) and macro sensitivity (right) of validation dataset for Linear SVM

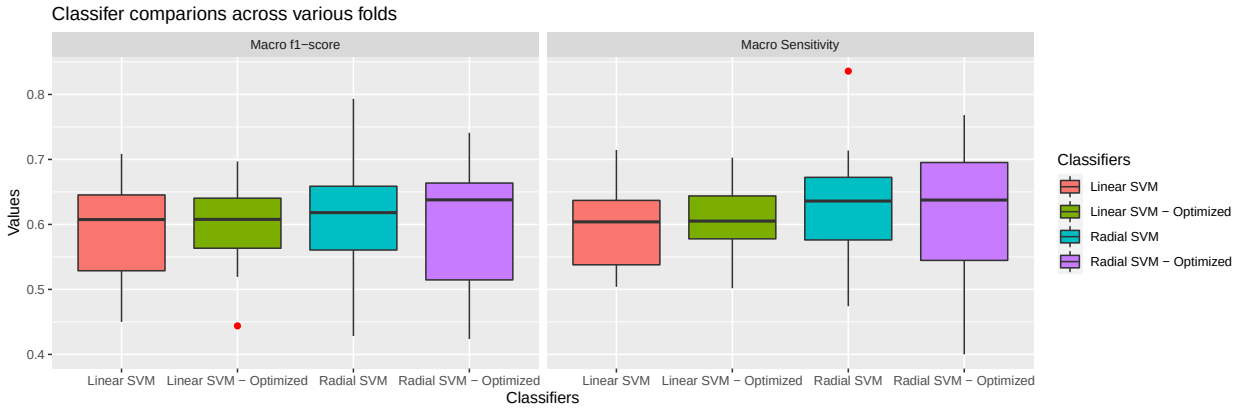
performed on principal componets after excluding correalted features for various values of cost. Linear SVM seems to be performing better for cost = 3.2 as it has high macro f1-score and high macro sensitivity. The code for this is in appendix.

We will perform similar steps for optimizing hyperparamters for radial SVM, but we will repeat the process for various values of cost i.e. 100, 200 to 1000 and gamma i.e. 0.001,0.002 to 0.01. The code for this is available in appendix.

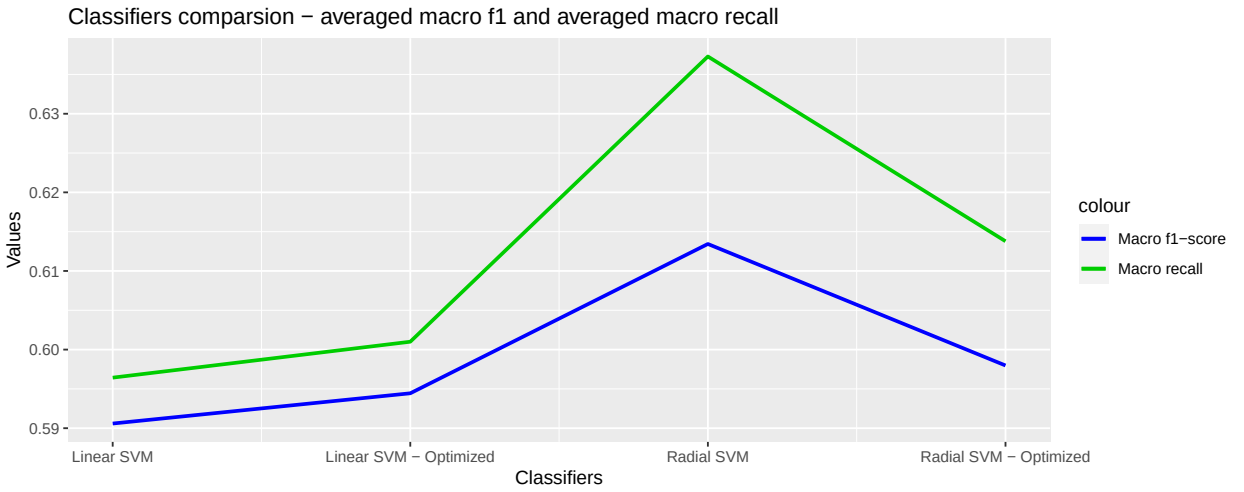


These plots indicate macro f1-score (left) and macro sensitivity (right) of validation dataset for radial SVM performed on principal componets after excluding correalted features for various values of cost and gamma. These plots are not the entire analysis but only a representation of required results. Radial SVM seems to be performing better for cost = 800 and gamma = 0.009. The code for this is available in appendix.

We will now compare Linear SVM, Radial SVM with default hyperparamters and with optimized hyperparameters. The code for this is available in appendix.



The boxplot is a comparison macro f1, macro recall of Linear SVM, Linear SVM with optimized cost, Radial SVM and Radial sVM with optimized cost and gamma across various folds on validation dataset. It appears that Linear SVM with optimized cost and Radial SVM with default parameters have less variability across different fold of data and also have similar performance. The code for this is available in appendix.



The line graph is a comparison of average of macro f1 and macro recall over different folds of data and this indicates that Radial SVM with default parameters has a better average macro f1 and better average macro recall over others i.e. 0.61, 0.63. The code for this is available in appendix.

Multi-class area under the roc curve for radial SVM with default paramters on validation dataset is 0.7471. This indicates the model is good in distinguishing between classes but is not perfect.

Conclusion

EDA suggested that the classes are not balanced, the dataset has high dimensions and 13 features are correlated. Further evaluation concluded that removing correlated feature, applying PCA, oversampling of minor classes and using radial SVM with default cost i.e. 1 and default gamma i.e. 0.02 provided better results and can be considered a reasonable approach. Dropping the correlated features resulted in less variability of performance across different folds of data. Feature subset did not help in improving the performance and also performing feature subset for polynomial SVM was computationally inefficient and required a lot of time and hence only a few iterations for polynomial SVM feature subset could be performed, perhaps performing feature subset for polynomial SVM in a high computation environment might yield some interesting results.

The dataset also does not have samples for 3 of 16 classes and this is preventing our model to make more accurate predictions. The dataset should also include more samples for some classes so that our classifiers does not rely on oversampling techniques.

This can be good starting point for us to use this approach to reduce false alerts in the clinical areas and assist clinical staff but this needs some more work i.e. more gold standard data, more computation efficient environment to explore other strategies.

References

Arrhythmia Data Set at: http://archive.ics.uci.edu/ml/datasets/Arrhythmia

Krzysztof, G 2019, Reducing False Arrhythmia Alarms at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6479538/

Christopher, G 2019, Handling Multi-Class Imbalance at: https://towardsdatascience.com/multi-class-imbalance-154ef2b15816

Appendix

The below is the code for creating histogram of labels.

```
library(ggplot2)
ggplot(data = arrhythmia_df, aes(x = class)) +
  geom_histogram(
    data = subset(arrhythmia_df, class == 1),
    fill = "forestgreen",
    alpha = 0.8
  ) +
  geom_histogram(
    data = subset(arrhythmia_df, class != 1),
    fill = "firebrick",
    alpha = 0.8
  ) +
  ggtitle("Frequency of labels") +
  xlab("Class") + ylab("Count") +
  geom_text(
    stat = 'count',
    mapping = aes(label = paste(round((..count..) / sum(..count..) * 100
    ), "%", sep = "")),
    position = position_stack(vjust = 1.05),
    size = 4
  ) +
  scale_x_continuous(breaks = seq(0, max(arrhythmia_df$class), 1))  +
  theme_classic()
```

The below is the code for creating barplot of missing values.

```
library(skimr)
summary_df = skim(arrhythmia_df) #We will use for missing values
missing_stats = data.frame(summary_df[summary_df$n_missing != 0, c("skim_variable", "n_missing")]) #Onl
colnames(missing_stats) = c("column_name", "no_of_missing")
missing_stats$percent_missing = paste(round(missing_stats$no_of_missing /
                                              nrow(arrhythmia_df) * 100), "%") #Percenttage calculation
library(ggplot2)
ggplot(data = missing_stats, aes(x = column_name, y = no_of_missing)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  ggtitle("Missing observations") +
  xlab("Column Names") + ylab("Missing observations") + ylim(c(0, nrow(arrhythmia_df))) +
  geom_text(aes(label = percent_missing)) + theme_classic()
```

The below is the code for box plot of height and weight.

```
require(gridExtra)
plot1 = ggplot(data = arrhythmia_df_nolabels, aes(x = Height)) +
  geom_boxplot(
    outlier.colour = "red",
    outlier.shape = 16,
    outlier.size = 2,
    notch = FALSE
  ) +  ggtitle("Spread of Height")
plot2 = ggplot(data = arrhythmia_df_nolabels, aes(x = Weight)) +
  geom_boxplot(
    outlier.colour = "red",
    outlier.shape = 16,
```

```
    outlier.size = 2,
    notch = FALSE
  ) +  ggtitle("Spread of Weight")
grid.arrange(plot1, plot2, ncol = 2)
```

The below is the code for box plot of V101,V201 and V301.

```
#Side by side boxplot for similar features for better comparison
Columns_for_boxplot = c("V101", "V201", "V301")
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("Vx01", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = Vx01)) +
  geom_boxplot(
    outlier.colour = "red",
    outlier.shape = 16,
    outlier.size = 2,
    notch = FALSE
  ) + ggtitle("Spread of V101,V201,V301") + xlab("Value") + theme(axis.text =
                                                                    element_text(size = 14),
                                                                  axis.title = element_text(size = 14))
```

The below is the code for box plot of DIII188 and DIII189.

```
Columns_for_boxplot = c("DIII188", "DIII189")  #Side by side boxplot for similar features for better co
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("DIII18x", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = DIII18x)) +
  geom_boxplot(
    outlier.colour = "red",
    outlier.shape = 16,
    outlier.size = 2,
    notch = FALSE
  ) + ggtitle("Spread of DIII188,DIII189") + xlab("Value") + theme(axis.text =
                                                                     element_text(size = 14),
                                                                   axis.title = element_text(size = 14))
```

The below is the code used for examining all remaining features.

```
Columns_for_boxplot = c("Q_Wave", "R_Wave", "S_Wave")
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of width") + xlab("Milli seconds")
#Can consider later
Columns_for_boxplot = c("R'_Wave", "S'_Wave", "Int_Def", "Rag_R_Nom")
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
```

```
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of peaks after R") + xlab("Milli seconds")
Columns_for_boxplot = c("Diph_R_Nom",
                        "Rag_P_Nom",
                        "Diph_P_Nom",
                        "Rag_T_Nom",
                        "Diph_T_Nom")
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of Norm") + xlab("Milli seconds")
Columns_for_boxplot = c(
  "DII00",
  "DII01",
  "DII02",
  "DII03",
  "DII04",
  "DII05",
  "DII06",
  "DII07",
  "DII08",
  "DII09",
  "DII10",
  "DII11",
  "DIII00",
  "DIII01",
  "DIII02",
  "DIII03",
  "DIII04",
  "DIII05",
  "DIII06",
  "DIII07",
  "DIII08",
  "DIII09",
  "DIII10",
  "DIII11"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                          element_text(s
```

```
                                                                                axis.title = elem
Columns_for_boxplot = c(
  "DII00",
  "DII01",
  "DII02",
  "DII03",
  "DII04",
  "DII05",
  "DII06",
  "DII07",
  "DII08",
  "DII09",
  "DII10",
  "DII11",
  "DIII00",
  "DIII01",
  "DIII02",
  "DIII03",
  "DIII04",
  "DIII05",
  "DIII06",
  "DIII07",
  "DIII08",
  "DIII09",
  "DIII10",
  "DIII11"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
Columns_for_boxplot = c(
  "AVR00",
  "AVR01",
  "AVR02",
  "AVR03",
  "AVR04",
  "AVR05",
  "AVR06",
  "AVR07",
  "AVR08",
  "AVR09",
  "AVR10",
  "AVR11",
  "AVL00",
  "AVL01",
  "AVL02",
  "AVL03",
  "AVL04",
  "AVL05",
  "AVL06",
  "AVL07",
  "AVL08",
  "AVL09",
  "AVL10",
```

```
  "AVL11"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                          element_text(si
                                                                                        axis.title = elem
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                          element_text(si
                                                                                        axis.title = elem
Columns_for_boxplot = c(
  "AVF00",
  "AVF01",
  "AVF02",
  "AVF03",
  "AVF04",
  "AVF05",
  "AVF06",
  "AVF07",
  "AVF08",
  "AVF09",
  "AVF10",
  "AVF11"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip()
Columns_for_boxplot = c(
  "V100",
  "V101",
  "V102",
  "V103",
  "V104",
  "V105",
  "V106",
  "V107",
  "V108",
  "V109",
```

```
  "V110",
  "V111",
  "V200",
  "V201",
  "V202",
  "V203",
  "V204",
  "V205",
  "V206",
  "V207",
  "V208",
  "V209",
  "V210",
  "V211"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of ............") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                    element_text(s
                                                                                  axis.title = ele

Columns_for_boxplot = c(
  "V300",
  "V301",
  "V302",
  "V303",
  "V304",
  "V305",
  "V306",
  "V307",
  "V308",
  "V309",
  "V310",
  "V311",
  "V400",
  "V401",
  "V402",
  "V403",
  "V404",
  "V405",
  "V406",
  "V407",
  "V408",
  "V409",
  "V410",
  "V411"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
```

```
Columns_for_boxplot = c("V101", "V201", "V301")
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("Vx01", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = Vx01)) +
  geom_boxplot(
    outlier.colour = "red",
    outlier.shape = 16,
    outlier.size = 2,
    notch = FALSE
  ) + ggtitle("Spread of V101,V201,V301") + xlab("Value") + theme(axis.text =
                                                                    element_text(size = 14),
                                                                  axis.title = element_text(size = 14))
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                          element_text(s
                                                                                        axis.title = ele

Columns_for_boxplot = c(
  "V500",
  "V501",
  "V502",
  "V503",
  "V504",
  "V505",
  "V506",
  "V507",
  "V508",
  "V509",
  "V510",
  "V511",
  "V600",
  "V601",
  "V602",
  "V603",
  "V604",
  "V605",
  "V606",
  "V607",
  "V608",
  "V609",
  "V610",
  "V611"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
```

```
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                      element_text(si
                                                                                    axis.title = elem
Columns_for_boxplot = c(
  "JJ_Wave",
  "Amp_Q_Wave",
  "Amp_R_Wave",
  "Amp_S_Wave",
  "R_Prime_Wave",
  "S_Prime_Wave",
  "P_Wave",
  "T_Wave"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                      element_text(si
                                                                                    axis.title = elem
Columns_for_boxplot = c(
  "QRSA",
  "QRSTA",
  "DII170",
  "DII171",
  "DII172",
  "DII173",
  "DII174",
  "DII175",
  "DII176",
  "DII177",
  "DII178",
  "DII179"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                      element_text(si
                                                                                    axis.title = elem
Columns_for_boxplot = c(
  "DIII180",
  "DIII181",
  "DIII182",
  "DIII183",
  "DIII184",
```

```
  "DIII185",
  "DIII186",
  "DIII187",
  "DIII188",
  "DIII189"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                          element_text(si
                                                                                        axis.title = elem
Columns_for_boxplot = c(
  "AVR190",
  "AVR191",
  "AVR192",
  "AVR193",
  "AVR194",
  "AVR195",
  "AVR196",
  "AVR197",
  "AVR198",
  "AVR199"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                          element_text(si
                                                                                        axis.title = elem
Columns_for_boxplot = c(
  "AVL200",
  "AVL201",
  "AVL202",
  "AVL203",
  "AVL204",
  "AVL205",
  "AVL206",
  "AVL207",
  "AVL208",
  "AVL209"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
```

```
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                          element_text(s
                                                                                       axis.title = ele
Columns_for_boxplot = c(
  "AVF210",
  "AVF211",
  "AVF212",
  "AVF213",
  "AVF214",
  "AVF215",
  "AVF216",
  "AVF217",
  "AVF218",
  "AVF219"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                          element_text(s
                                                                                       axis.title = ele
Columns_for_boxplot = c(
  "V1220",
  "V1221",
  "V1222",
  "V1223",
  "V1224",
  "V1225",
  "V1226",
  "V1227",
  "V1228",
  "V1229",
  "V2230",
  "V2231",
  "V2232",
  "V2233",
  "V2234",
  "V2235",
  "V2236",
  "V2237",
  "V2238",
  "V2239"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
```

```
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                          element_text(s
                                                                                        axis.title = ele
Columns_for_boxplot = c(
  "V3240",
  "V3241",
  "V3242",
  "V3243",
  "V3244",
  "V3245",
  "V3246",
  "V3247",
  "V3248",
  "V3249",
  "V4250",
  "V4251",
  "V4252",
  "V4253",
  "V4254",
  "V4255",
  "V4256",
  "V4257",
  "V4258",
  "V4259"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                          element_text(s
                                                                                        axis.title = ele

Columns_for_boxplot = c(
  "V5260",
  "V5261",
  "V5262",
  "V5263",
  "V5264",
  "V5265",
  "V5266",
  "V5267",
  "V5268",
  "V5269",
  "V6270",
  "V6271",
```

```
  "V6272",
  "V6273",
  "V6274",
  "V6275",
  "V6276",
  "V6277",
  "V6278",
  "V6279"
)
wfdur_df = data.frame(rep(Columns_for_boxplot, times = nrow(arrhythmia_df)), as.vector(t(arrhythmia_df[
colnames(wfdur_df) = c("WaveForm", "Value")
ggplot(data = wfdur_df, aes(x = Value, y = WaveForm)) +   geom_boxplot(
  outlier.colour = "red",
  outlier.shape = 16,
  outlier.size = 2,
  notch = FALSE
) + ggtitle("Spread of DII08 and others") + xlab("Milli seconds") + coord_flip() +  theme(axis.text =
                                                                                          element_text(s
                                                                                        axis.title = ele
```

The below is the code for visualizing continous correlated ECG features

```
#The belo function is to create a smooting line in the ggpairs plot
regline_fn <- function(data, mapping, ...) {
  p <- ggplot(data = data, mapping = mapping) + geom_point() +
    geom_smooth(method = loess,
                fill = "red",
                color = "red",...) +
    geom_smooth(method = lm,
                fill = "blue",
                color = "blue",...)
  p
}
require(gridExtra)
plot1 = ggpairs(processed_arrhythmia_df_nolabels[, c(related_numerical_cols[1, 1], related_numerical_co
                lower = list(continuous = regline_fn))
plot2 = ggpairs(processed_arrhythmia_df_nolabels[, c(related_numerical_cols[2, 1], related_numerical_co
                lower = list(continuous = regline_fn))
plot3 = ggpairs(processed_arrhythmia_df_nolabels[, c(related_numerical_cols[3, 1], related_numerical_co
                lower = list(continuous = regline_fn))
plot4 = ggpairs(processed_arrhythmia_df_nolabels[, c(related_numerical_cols[4, 1], related_numerical_co
                lower = list(continuous = regline_fn))
grid.arrange(
  grid.grabExpr(print(plot1)),
  grid.grabExpr(print(plot2)),
  grid.grabExpr(print(plot3)),
  grid.grabExpr(print(plot4)),
  ncol = 4
)
```

The below is the Code for performing PCA on dataset with correlated features and without correlated features.

```
arrhythmia.pca <- prcomp(processed_arrhythmia_df_nolabels) #PCA of dataset with correlated columns
tot.var <- sum(arrhythmia.pca$sdev ^ 2)
var.explained <-  data.frame(pc = seq(1:278),
            var.explained  = arrhythmia.pca$sdev ^ 2 / tot.var)
cum_var.explained = c()
for (i in seq(nrow(var.explained)))
{
  cum_var.explained =  c(cum_var.explained, sum(var.explained[1:i, 2]) * 100)  #Calculating cumulative
}

processed_arrhythmia_df_nolabels2 = processed_arrhythmia_df_nolabels[, -which(names(arrhythmia_df_nolabe
arrhythmia.pca2 <- prcomp(processed_arrhythmia_df_nolabels2) #PCA of dataset without correlated columns
tot.var <- sum(arrhythmia.pca2$sdev ^ 2)
var.explained <-
  data.frame(pc = seq(1:ncol(processed_arrhythmia_df_nolabels2)),
            var.explained  = arrhythmia.pca2$sdev ^ 2 / tot.var)
cum_var.explained2 = c()
for (i in seq(nrow(var.explained)))
{
  cum_var.explained2 =  c(cum_var.explained2, sum(var.explained[1:i, 2]) * 100) #Calculating cumulative
}
```

The below is the code for the plot showing cumulative variance explained by PCA

```
cum_var.explained_df <- data.frame(pc = seq(1:ncol(processed_arrhythmia_df_nolabels)),  cum_var.explain

pca1plot = ggplot(cum_var.explained_df[1:45, ], aes(pc, cum_var.explained)) + geom_bar(stat = "identity
  geom_hline(aes(yintercept = 95), colour = "red") + geom_text(aes(0, 95, label = "95% variability", vj
  labs(y = "% of cum. variance exp.") +
  scale_x_continuous(breaks = seq(0, 45), "Principal Components")  +
  theme_classic()

cum_var.explained_df2 <- data.frame(pc = seq(1:ncol(processed_arrhythmia_df_nolabels2)),  cum_var.expla

pca2plot = ggplot(cum_var.explained_df2[1:45, ], aes(pc, cum_var.explained2)) + geom_bar(stat = "identi
  geom_hline(aes(yintercept = 95), colour = "red") + geom_text(aes(0, 95, label = "95% variability", vj
  labs(y = "% of cum. variance exp.") +
  scale_x_continuous(breaks = seq(0, 45), "Principal Components")  +
  theme_classic()

require(gridExtra)
grid.arrange(pca1plot, pca2plot, nrow = 2)
```

The below code is for feature subset for linear SVM.

```
selectFeature <-
  function(train, test, cls.train, cls.test, features) {
    ## identify a feature to be selected
    current.best.fone <- -Inf
    selected.i <- NULL
    for (i in 1:ncol(train)) {
      current.f <- colnames(train)[i]
      if (!current.f %in% features) {
```

```
        subfeatures_df_train = data.frame(class = cls.train)
        subfeatures_df_test = data.frame(class = cls.test)
        allcols = c(features, current.f)
        for (eachcol in allcols)
        {
          subfeatures_df_train[, eachcol] = train[, eachcol]
          subfeatures_df_test[, eachcol] = test[, eachcol]
        }
        oversampled_df = subfeatures_df_train
        all_exis_classes  = sort(unique(arrhythmia_labels))

        for (eachminroclass in all_exis_classes[!all_exis_classes %in% 1])
        {
          testdf = subset(oversampled_df, class == eachminroclass |
                            class == 1)
          balanceddata <-
            ovun.sample(
              class ~ .,
              data = get("testdf", sys.frame(1)),
              method = "over",
              p = 0.5
            )$data
          oversampled_df = rbind(
            subset(balanceddata, class != 1),
            subset(oversampled_df, class != eachminroclass)
          )
        }
        linsvm.model1 <-
          svm(
            x = oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop =
                               FALSE],
            y = oversampled_df[, "class"],
            kernel = "linear",
            type = "C-classification"
          )
        preds <-
          predict(linsvm.model1, newdata = subfeatures_df_test[, -which(colnames(subfeatures_df_test) %
                                                                 FALSE])
        f1score <- calcF1Scores(as.numeric(levels(preds))[preds], cls.test)
        test.fone = round(f1score[[1]], 3)
        if (test.fone >  current.best.fone) {
          current.best.fone <- test.fone
          selected.i <- colnames(train)[i]
        }
      }
    }
    return(list(selected.i, current.best.fone))
  }

library(caret)
set.seed(1)
inTrain <- createDataPartition(arrhythmia_labels, p = .6)[[1]]
allFeatures <- colnames(processed_arrhythmia_df_nolabels)
```

```
train <- processed_arrhythmia_df_nolabels[inTrain, ]
test  <- processed_arrhythmia_df_nolabels[-inTrain, ]
cls.train <- arrhythmia_labels[inTrain]
cls.test <- arrhythmia_labels[-inTrain]

features <- NULL
fone_scores = NULL
# select the 1 to 30 best features using a wrapper classifier
for (j in 1:30) {
  featureoutput <-
    selectFeature(train, test, cls.train, cls.test, features)
  features <- c(features, featureoutput[[1]])
  fone_scores = c(fone_scores, featureoutput[[2]])
}
```

The below code is for feature subset for radial SVM.

```
selectFeature <-
  function(train, test, cls.train, cls.test, features) {
    ## identify a feature to be selected
    current.best.fone <- -Inf
    selected.i <- NULL
    for (i in 1:ncol(train)) {
      current.f <- colnames(train)[i]
      if (!current.f %in% features) {
        subfeatures_df_train = data.frame(class = cls.train)
        subfeatures_df_test = data.frame(class = cls.test)
        allcols = c(features, current.f)
        for (eachcol in allcols)
        {
          subfeatures_df_train[, eachcol] = train[, eachcol]
          subfeatures_df_test[, eachcol] = test[, eachcol]
        }

        oversampled_df = subfeatures_df_train
        all_exis_classes  = sort(unique(arrhythmia_labels))
        for (eachminroclass in all_exis_classes[!all_exis_classes %in% 1])
        {
          testdf = subset(oversampled_df, class == eachminroclass |
                            class == 1)
          balanceddata <-
            ovun.sample(
              class ~ .,
              data = get("testdf", sys.frame(1)),
              method = "over",
              p = 0.5
            )$data
          oversampled_df = rbind(
            subset(balanceddata, class != 1),
            subset(oversampled_df, class != eachminroclass)
          )
        }
```

```
        radial.model1 <-
          svm(
            x = oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop =
                               FALSE],
            y = oversampled_df[, "class"],
            kernel = "radial",
            type = "C-classification"
          )
        preds <-
          predict(radial.model1, newdata = subfeatures_df_test[, -which(colnames(subfeatures_df_test) %
                                                                         FALSE])

        f1score <- calcF1Scores(as.numeric(levels(preds))[preds], cls.test)
        test.fone = round(f1score[[1]], 3)
        #test.fone
        if (test.fone >  current.best.fone) {
          current.best.fone <- test.fone
          #fone_of_best = modelval.fone
          selected.i <- colnames(train)[i]
        }
      }
    }
    return(list(selected.i, current.best.fone))
  }

library(caret)
set.seed(1)
inTrain <- createDataPartition(arrhythmia_labels, p = .6)[[1]]
allFeatures <- colnames(processed_arrhythmia_df_nolabels)

train <- processed_arrhythmia_df_nolabels[inTrain, ]
test  <- processed_arrhythmia_df_nolabels[-inTrain, ]
cls.train <- arrhythmia_labels[inTrain]
cls.test <- arrhythmia_labels[-inTrain]

features <- NULL
fone_scores = NULL
for (j in 1:20) {
  featureoutput <-
    selectFeature(train, test, cls.train, cls.test, features)
  features <- c(features, featureoutput[[1]])
  fone_scores = c(fone_scores, featureoutput[[2]])
}
```

The below code is for feature subset for polynomial SVM.

```
selectFeature <-
  function(train, test, cls.train, cls.test, features) {
    current.best.fone <- -Inf
    selected.i <- NULL
    for (i in 1:ncol(train)) {
      current.f <- colnames(train)[i]
      if (!current.f %in% features) {
        subfeatures_df_train = data.frame(class = cls.train)
```

```
        subfeatures_df_test = data.frame(class = cls.test)
        allcols = c(features, current.f)
        for (eachcol in allcols)
        {
          subfeatures_df_train[, eachcol] = train[, eachcol]
          subfeatures_df_test[, eachcol] = test[, eachcol]
        }

        polynomial.model1 <-
          svm(
            x = subfeatures_df_train[, -which(colnames(subfeatures_df_train) %in% c("class")), drop =
                                       FALSE],
            y = subfeatures_df_train[, "class"],
            kernel = "polynomial",
            type = "C-classification",
            degree = 2
          )
        preds <-
          predict(polynomial.model1, newdata = subfeatures_df_test[, -which(colnames(subfeatures_df_tes
                                                                             FALSE])

        f1score <- calcF1Scores(as.numeric(levels(preds))[preds], cls.test)

        test.fone = round(f1score[[1]], 3)
        if (test.fone >  current.best.fone) {
          current.best.fone <- test.fone
          selected.i <- colnames(train)[i]
        }
      }
    }
    return(list(selected.i, current.best.fone))
  }

library(caret)
set.seed(1)
inTrain <- createDataPartition(arrhythmia_labels, p = .6)[[1]]
allFeatures <- colnames(processed_arrhythmia_df_nolabels)
train <- processed_arrhythmia_df_nolabels[inTrain, ]
oversampled_df = train
oversampled_df$class = arrhythmia_labels[inTrain]
all_exis_classes  = sort(unique(arrhythmia_labels))
for (eachminroclass in all_exis_classes[!all_exis_classes %in% 1])
{
  balanceddata <-
    ovun.sample(
      class ~ .,
      data = subset(oversampled_df, class == eachminroclass |
                      class == 1),
      method = "over",
      p = 0.5
    )$data
  oversampled_df = rbind(subset(balanceddata, class != 1),
                         subset(oversampled_df, class != eachminroclass))
```

```
}

train = oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop =
                         FALSE]
test  <- processed_arrhythmia_df_nolabels[-inTrain, ]
cls.train <- oversampled_df[, "class"]
cls.test <- arrhythmia_labels[-inTrain]

features = NULL
fone_scores = NULL

for (j in 1:30) {
  featureoutput <-
    selectFeature(train, test, cls.train, cls.test, features)
  features <- c(features, featureoutput[[1]])
  fone_scores = c(fone_scores, featureoutput[[2]])
}
```

The below code is for feature subset for Multinomial Log.

```
selectFeature <-
  function(train, test, cls.train, cls.test, features) {
    ## identify a feature to be selected
    current.best.fone <- -Inf
    selected.i <- NULL
    for (i in 1:ncol(train)) {
      current.f <- colnames(train)[i]
      if (!current.f %in% features) {
        subfeatures_df_train = data.frame(class = cls.train)
        subfeatures_df_test = data.frame(class = cls.test)
        allcols = c(features, current.f)
        for (eachcol in allcols)
        {
          subfeatures_df_train[, eachcol] = train[, eachcol]
          subfeatures_df_test[, eachcol] = test[, eachcol]
        }

        multinomlogmodel <-
          multinom(class ~ ., data = subfeatures_df_train, trace = FALSE)
        preds = predict(multinomlogmodel, newdata = subfeatures_df_test[, -which(colnames(subfeatures_d
                                                                                      FALSE])
        f1score <- calcF1Scores(as.numeric(levels(preds))[preds], cls.test)
        test.fone = round(f1score[[1]], 3)
        if (test.fone >  current.best.fone) {
          current.best.fone <- test.fone
          selected.i <- colnames(train)[i]
        }
      }
    }
    return(list(selected.i, current.best.fone))
  }

library(caret)
```

```
set.seed(1)
inTrain <- createDataPartition(arrhythmia_labels, p = .6)[[1]]
allFeatures <- colnames(processed_arrhythmia_df_nolabels)
train <- processed_arrhythmia_df_nolabels[inTrain, ]
oversampled_df = train
oversampled_df$class = arrhythmia_labels[inTrain]
all_exis_classes  = sort(unique(arrhythmia_labels))
for (eachminroclass in all_exis_classes[!all_exis_classes %in% 1])
{
  balanceddata <-
    ovun.sample(
      class ~ .,
      data = subset(oversampled_df, class == eachminroclass |
                      class == 1),
      method = "over",
      p = 0.5
    )$data
  oversampled_df = rbind(subset(balanceddata, class != 1),
                         subset(oversampled_df, class != eachminroclass))
}

train = oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop =
                         FALSE]
test  <- processed_arrhythmia_df_nolabels[-inTrain, ]
cls.train <- oversampled_df[, "class"]
cls.test <- arrhythmia_labels[-inTrain]

features = NULL
fone_scores = NULL
for (j in 1:20) {
  featureoutput <-
    selectFeature(train, test, cls.train, cls.test, features)
  features <- c(features, featureoutput[[1]])
  fone_scores = c(fone_scores, featureoutput[[2]])
}
```

The below code is for feature subset for LDA.

```
selectFeature <-
  function(train, test, cls.train, cls.test, features) {
    ## identify a feature to be selected
    current.best.fone <- -Inf
    selected.i <- NULL
    for (i in 1:ncol(train)) {
      current.f <- colnames(train)[i]
      if (!current.f %in% features) {
        subfeatures_df_train = data.frame(class = cls.train)
        subfeatures_df_test = data.frame(class = cls.test)
        allcols = c(features, current.f)
        for (eachcol in allcols)
        {
          subfeatures_df_train[, eachcol] = train[, eachcol]
          subfeatures_df_test[, eachcol] = test[, eachcol]
```

```
        }
        flag <- TRUE
        tryCatch({
          lda.model <- MASS::lda(class ~ ., data = subfeatures_df_train)
        }, error = function(e) {
          flag <<- FALSE
        })

        if (!flag) {
          next
        }
        ldapreds = predict(lda.model, newdata = subfeatures_df_test[, -which(colnames(subfeatures_df_tes
                                                                              FALSE])$class
        f1score <-
          calcF1Scores(as.numeric(levels(ldapreds))[ldapreds], cls.test)
        test.fone = round(f1score[[1]], 3)
        if (test.fone >  current.best.fone) {
          current.best.fone <- test.fone
          selected.i <- colnames(train)[i]
        }
      }
    }
    return(list(selected.i, current.best.fone))
  }

library(caret)
set.seed(1)
inTrain <- createDataPartition(arrhythmia_labels, p = .6)[[1]]
allFeatures <- colnames(processed_arrhythmia_df_nolabels)
train <- processed_arrhythmia_df_nolabels[inTrain, ]
oversampled_df = train
oversampled_df$class = arrhythmia_labels[inTrain]
all_exis_classes  = sort(unique(arrhythmia_labels))
for (eachminroclass in all_exis_classes[!all_exis_classes %in% 1])
{
  balanceddata <-
    ovun.sample(
      class ~ .,
      data = subset(oversampled_df, class == eachminroclass |
                      class == 1),
      method = "over",
      p = 0.5
    )$data
  oversampled_df = rbind(subset(balanceddata, class != 1),
                         subset(oversampled_df, class != eachminroclass))
}

train = oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop =
                         FALSE]
test  <- processed_arrhythmia_df_nolabels[-inTrain, ]
cls.train <- oversampled_df[, "class"]
cls.test <- arrhythmia_labels[-inTrain]
```

```
features = NULL
fone_scores = NULL
for (j in 1:30) {
  featureoutput <-
    selectFeature(train, test, cls.train, cls.test, features)
  features <- c(features, featureoutput[[1]])
  fone_scores = c(fone_scores, featureoutput[[2]])
}
```

The below code is to visualize feature subset performance in various subplots .

```
p1= ggplot(data=knn_featuresubset_metrics ,aes(x=index,y=macrof1))+geom_line(data=knn_featuresubset_met

p2= ggplot(data=linsvm_featuresubset_metrics ,aes(x=index,y=macrof1))+geom_line(data=linsvm_featuresubs

p3= ggplot(data=radsvm_featuresubset_metrics,aes(x=index,y=macrof1))+geom_line(data=radsvm_featuresubse

p4= ggplot(data=multilog_featuresubset_metrics ,aes(x=index,y=macrof1))+geom_line(data=multilog_feature

p5= ggplot(data=lda_featuresubset_metrics ,aes(x=index,y=macrof1))+geom_line(data=lda_featuresubset_met

p6= ggplot(data=polysvm_featuresubset_metrics ,aes(x=index,y=macrof1))+geom_line(data=polysvm_featuresu

require(gridExtra)

grid.arrange(p1,p2,p3,p4,p5,p6, ncol=3,nrow=2)
```

The below is the code for approach 2 i.e. using dataset with correlated features and then applying PCA to evaluate classifers.

```
set.seed(1)
fold <- createFolds(arrhythmia_labels, k = 10) # apply 10-fold cross-validation
tmp2f1knn = tmp2f1linsvm = tmp2f1ploy = tmp2f1radc = tmp2f1lda = tmp2f1log = c()
tmp2recaknn = tmp2recalinsvm = tmp2recaploy = tmp2recaradc = tmp2recalda = tmp2recalog = c()

for (i in 1:length(fold)) {
  arrhythmia.pca <-
    prcomp(processed_arrhythmia_df_nolabels[-fold[[i]], -which(names(processed_arrhythmia_df_nolabels)
  tot.var <- sum(arrhythmia.pca$sdev ^ 2)
  var.explained <- data.frame(pc = seq(1:ncol(processed_arrhythmia_df_nolabels[-fold[[i]], -which(names

  cum_var.explained = c()
  for (u in seq(nrow(var.explained)))
  {
    cum_var.explained =  c(cum_var.explained, sum(var.explained[1:u, 2]) * 100)
  }
  #identify minimum PCs for 95% variability

  pctouse = min(which(cum_var.explained > 95))

  #Oversample minor classes in training data
  oversampled_df = data.frame(arrhythmia.pca$x[, 1:pctouse])
  oversampled_df$class = arrhythmia_labels[-fold[[i]]]
```

```
all_exis_classes  = sort(unique(arrhythmia_labels))

for (eachminroclass in all_exis_classes[!all_exis_classes %in% 1])
{
  balanceddata <-
    ovun.sample(
      class ~ .,
      data = subset(oversampled_df, class == eachminroclass |
                      class == 1),
      method = "over",
      p = 0.5
    )$data

  oversampled_df = rbind(subset(balanceddata, class != 1),
                         subset(oversampled_df, class != eachminroclass))
}
oversampled_df <- oversampled_df[sample(nrow(oversampled_df)), ]  #Shuffling rows

#apply PCA on validation data
arrhythmia.pcatest = predict(arrhythmia.pca, processed_arrhythmia_df_nolabels[fold[[i]], -which(names

#Train classifiers
preds <- knn(oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop =FALSE],data.fra
f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
tmp2f1knn = c(tmp2f1knn, f1score[[1]])
tmp2recaknn = c(tmp2recaknn, f1score[[2]])

linsvm.model1 <- svm(x = oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop =FALS
preds <- predict(linsvm.model1, newdata = data.frame(arrhythmia.pcatest[, 1:pctouse]))
f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
tmp2f1linsvm = c(tmp2f1linsvm, f1score[[1]])
tmp2recalinsvm = c(tmp2recalinsvm, f1score[[2]])


polysvm.model1 <- svm(x = oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop =FA
preds <- predict(polysvm.model1, newdata = data.frame(arrhythmia.pcatest[, 1:pctouse]))
f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
tmp2f1ploy = c(tmp2f1ploy, f1score[[1]])
tmp2recaploy = c(tmp2recaploy, f1score[[2]])

radsvm.model1 <- svm(x = oversampled_df[, -which(colnames(oversampled_df) %in% c("class")), drop =FALS
preds <- predict(radsvm.model1, newdata = data.frame(arrhythmia.pcatest[, 1:pctouse]))
f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
tmp2f1radc = c(tmp2f1radc, f1score[[1]])
tmp2recaradc = c(tmp2recaradc, f1score[[2]])

lda.model <- MASS::lda(class ~ ., data = oversampled_df)
preds = predict(lda.model, newdata = data.frame(arrhythmia.pcatest[, 1:pctouse]))$class
f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
tmp2f1lda = c(tmp2f1lda, f1score[[1]])
tmp2recalda = c(tmp2recalda, f1score[[2]])

multinomlogmodel <- multinom(class ~ ., data = oversampled_df, trace = FALSE)
```

```
  preds = predict(multinomlogmodel, newdata = data.frame(arrhythmia.pcatest[, 1:pctouse]))
  f1score <- calcF1Scores(as.numeric(levels(preds))[preds], arrhythmia_labels[fold[[i]]])
  tmp2f1log = c(tmp2f1log, f1score[[1]])
  tmp2recalog = c(tmp2recalog, f1score[[2]])

}

fone.val = c(mean(tmp2f1log ),mean(tmp2f1lda),mean(tmp2f1knn),mean(tmp2f1linsvm),mean(tmp2f1ploy),mean(
recall.val = c(mean(tmp2recalog ),mean(tmp2recalda),mean(tmp2recaknn),mean(tmp2recalinsvm),mean(tmp2reca

plot_df95pca2 = data.frame(index= seq(1:6),fone=fone.val,recall=recall.val,algo= c("Multinomial Log","LI
    ggplot(data=plot_df95pca2 ,aes(x=index,y=fone))+geom_line(data=plot_df95pca2 ,aes(x=index,y=fone,col

boxplot_df_pca295= data.frame("Values"=c(tmp2recalog,tmp2recalda,tmp2recaknn,tmp2recalinsvm,tmp2recaploy

 ggplot(data=boxplot_df_pca295, aes(x=Classifiers,y=Values, fill=Classifiers)) +   geom_boxplot(outlier
```

The below code is for comparing classifers across different approaches via boxplot.

```
newdf = rbind(boxplot_df_pca95,boxplot_df_pca295,boxplot_df_subset)
newdf$dataset = rep(c("PCA including correlated features","PCA excluding correlated features","Feature S

 ggplot(data=newdf, aes(x=dataset,y=Values, fill=Classifiers)) +   geom_boxplot(outlier.colour="red", ou
```

The below code is for comparing classifers across different approaches via line plot.

```
ggplot(data=plot_df95pca2 ,aes(x=index,y=fone))+geom_line(data=plot_df95pca2 ,aes(x=index,y=fone,color=
```

The below code is for visualizing lineavr SVM performance for different values of cost - hyperparameter optimization

```
lin.plot.dat = subset(lin.plot.dat,k <5.1)
ggplot(lin.plot.dat) + geom_line(aes(x = k, y = values, fill= metric))+ scale_x_continuous(breaks = lin
```

The below code is for hyper paramter optimization of radial SVM.

```
fold <- createFolds(arrhythmia_labels, k=10)

knn.sen = knn.fone = c()
knn.fnr.k = c()

gamma_Values = seq(0.001,0.01,0.001)#c(0.001,0.01,0.1, 1, 10, 100)
cost_values =  seq(100,1000,100) #seq(10)

gamma_cost_values = cbind(rep(gamma_Values, times = length(cost_values)),rep(cost_values, each = length

for (idx in seq(nrow(gamma_cost_values)))
{
 knn.TP <- knn.TN <- knn.FP <- knn.FN <- c()
lda.TP <- lda.TN <- lda.FP <- lda.FN <- c()
f1scores = c()
```

```
sens_Scores = c()
for(i in 1:length(fold)){

arrhythmia.pca <- prcomp(processed_arrhythmia_df_nolabels[-fold[[i]],-which(names(processed_arrhythmia_
tot.var <- sum(arrhythmia.pca$sdev^2)
var.explained <- data.frame(pc = seq(1:ncol(processed_arrhythmia_df_nolabels[-fold[[i]],-which(names(pr
cum_var.explained =c()
for (u in seq(nrow(var.explained)))
{cum_var.explained =  c(cum_var.explained,sum(var.explained[1:u,2])*100)}

pctouse = min(which(cum_var.explained > 95))
oversampled_df = data.frame(arrhythmia.pca$x[,1:pctouse])
oversampled_df$class = arrhythmia_labels[-fold[[i]]]
all_exis_classes  = sort(unique(arrhythmia_labels))
for (eachminroclass in all_exis_classes[!all_exis_classes %in% 1])
{
balanceddata<-ovun.sample(class ~ ., data =subset(oversampled_df,class == eachminroclass | class == 1),
oversampled_df = rbind(subset(balanceddata,class != 1),subset(oversampled_df,class != eachminroclass))
}
oversampled_df <- oversampled_df[sample(nrow(oversampled_df)),]  #Shuffling rows
arrhythmia.pcatest = predict(arrhythmia.pca,processed_arrhythmia_df_nolabels[fold[[i]],-which(names(pro

 svm.model1 <- svm(x = oversampled_df[,-which(colnames(oversampled_df) %in% c("class")),drop=FALSE], y =
    preds <- predict(svm.model1, newdata=data.frame(arrhythmia.pcatest[,1:pctouse]) )
    f1score <- calcF1Scores(as.numeric(levels(preds))[preds],arrhythmia_labels[fold[[i]]])
  f1scores = c(f1scores,f1score[[1]])
    sens_Scores = c(sens_Scores,f1score[[2]])
}

knn.fone = c(knn.fone,mean(f1scores))
knn.sen = c(knn.sen,mean(sens_Scores))

}
radvm.plot.dat <- data.frame(values = c(unlist(knn.fone),unlist(knn.sen)),metric = rep(c("Macro f1-scor
```

The below code is for visualizing radial SVM performance for different values of cost and gamma - hyperparameter optimization

```
t1 = ggplot(data = subset(radvm.plot.dat , metric == "Macro f1-score" &
                            cost == 100) ,
            aes(x = gamma, y = values)) + geom_line(
              data = subset(radvm.plot.dat , metric == "Macro f1-score" &
                              cost == 100) ,
              aes(x = gamma, y = values, color = "Cost = 100") ,
              size = 1
            ) + geom_line(
              data = subset(radvm.plot.dat , metric == "Macro f1-score" &
                              cost == 800) ,
              aes(x = gamma, y = values, color = "Cost = 800") ,
              size = 1
            ) + geom_line(
              data = subset(radvm.plot.dat , metric == "Macro f1-score" &
                              cost == 900) ,
```

```
              aes(x = gamma, y = values, color = "Cost = 900") ,
              size = 1
            ) + labs(title = "Macro f1-score for various cost and gamma - Radial SVM", x =
                       "Gamma values", y = "Macro f1-score") + scale_x_continuous(breaks = radvm.plot.da
  scale_color_discrete(name = "Legend")
t2 = ggplot(
  data = subset(radvm.plot.dat , metric == "Macro sensitivity" &
                  cost == 100) ,
  aes(x = gamma, y = values)
) + geom_line(
  data = subset(radvm.plot.dat , metric == "Macro sensitivity" &
                  cost == 100) ,
  aes(x = gamma, y = values, color = "Cost = 100") ,
  size = 1
) + geom_line(
  data = subset(radvm.plot.dat , metric == "Macro sensitivity" &
                  cost == 800) ,
  aes(x = gamma, y = values, color = "Cost = 800") ,
  size = 1
) + geom_line(
  data = subset(radvm.plot.dat , metric == "Macro sensitivity" &
                  cost == 900) ,
  aes(x = gamma, y = values, color = "Cost = 900") ,
  size = 1
) + labs(title = "Macro sensitivity for various cost and gamma - Radial SVM", x =
           "Gamma values", y = "Macro sensitivity") + scale_x_continuous(breaks = radvm.plot.dat$gamma)
  scale_color_discrete(name = "Legend")
require(gridExtra)
grid.arrange(t1, t2, ncol = 2)
```

The below code is for training and comparing, linear SVM, radial SVM with default and optimized parameters and also for visualizing the results via box plot.

```
fold <- createFolds(arrhythmia_labels, k=10)
# apply 10-fold cross-validation

tmp2f1knn =tmp2f1linsvm =tmp2f1linsvmop= tmp2f1ploy = tmp2f1radc =tmp2f1radcop= tmp2f1lda = tmp2f1log =
tmp2recaknn =tmp2recalinsvm =tmp2recalinsvmop= tmp2recaploy = tmp2recaradc =tmp2recaradcop= tmp2recalda

for(i in 1:length(fold)){

arrhythmia.pca <- prcomp(processed_arrhythmia_df_nolabels[-fold[[i]],-which(names(processed_arrhythmia_
tot.var <- sum(arrhythmia.pca$sdev^2)
var.explained <- data.frame(pc = seq(1:ncol(processed_arrhythmia_df_nolabels[-fold[[i]],-which(names(pr

cum_var.explained =c()
for (u in seq(nrow(var.explained)))
{cum_var.explained =  c(cum_var.explained,sum(var.explained[1:u,2])*100)}

#Identifying the number of principal components to explain 95% var
pctouse = min(which(cum_var.explained > 95))

#Over sampling minor classes
```

```
oversampled_df = data.frame(arrhythmia.pca$x[,1:pctouse])
oversampled_df$class = arrhythmia_labels[-fold[[i]]]
all_exis_classes  = sort(unique(arrhythmia_labels))

for (eachminroclass in all_exis_classes[!all_exis_classes %in% 1])
{
balanceddata<-ovun.sample(class ~ ., data =subset(oversampled_df,class == eachminroclass | class == 1),
oversampled_df = rbind(subset(balanceddata,class != 1),subset(oversampled_df,class != eachminroclass))
}
oversampled_df <- oversampled_df[sample(nrow(oversampled_df)),]  #Shuffling rows

#apply PCA on test
arrhythmia.pcatest = predict(arrhythmia.pca,processed_arrhythmia_df_nolabels[fold[[i]],-which(names(pro

linsvm.model1 <- svm(x = oversampled_df[,-which(colnames(oversampled_df) %in% c("class")),drop=FALSE], y
preds <- predict(linsvm.model1, newdata=data.frame(arrhythmia.pcatest[,1:pctouse]))
f1score <- calcF1Scores(as.numeric(levels(preds))[preds],arrhythmia_labels[fold[[i]]])
tmp2f1linsvm = c(tmp2f1linsvm,f1score[[1]])
tmp2recalinsvm = c(tmp2recalinsvm,f1score[[2]])


linsvm.model1 <- svm(x = oversampled_df[,-which(colnames(oversampled_df) %in% c("class")),drop=FALSE], y
preds <- predict(linsvm.model1, newdata=data.frame(arrhythmia.pcatest[,1:pctouse]))
f1score <- calcF1Scores(as.numeric(levels(preds))[preds],arrhythmia_labels[fold[[i]]])
tmp2f1linsvmop = c(tmp2f1linsvmop,f1score[[1]])
tmp2recalinsvmop = c(tmp2recalinsvmop,f1score[[2]])

radsvm.model1 <- svm(x = oversampled_df[,-which(colnames(oversampled_df) %in% c("class")),drop=FALSE], y
preds <- predict(radsvm.model1, newdata=data.frame(arrhythmia.pcatest[,1:pctouse]))
f1score <- calcF1Scores(as.numeric(levels(preds))[preds],arrhythmia_labels[fold[[i]]])
tmp2f1radc = c(tmp2f1radc,f1score[[1]])
tmp2recaradc = c(tmp2recaradc,f1score[[2]])

radsvm.model1 <- svm(x = oversampled_df[,-which(colnames(oversampled_df) %in% c("class")),drop=FALSE], y
preds <- predict(radsvm.model1, newdata=data.frame(arrhythmia.pcatest[,1:pctouse]))
f1score <- calcF1Scores(as.numeric(levels(preds))[preds],arrhythmia_labels[fold[[i]]])
tmp2f1radcop = c(tmp2f1radcop,f1score[[1]])
tmp2recaradcop = c(tmp2recaradcop,f1score[[2]])
}

fone.val = c(mean(tmp2f1linsvm),mean(tmp2f1linsvmop),mean(tmp2f1radc),mean(tmp2f1radcop))
recall.val = c(mean(tmp2recalinsvm),mean(tmp2recalinsvmop),mean(tmp2recaradc),mean(tmp2recaradcop))

plot_df95pca2opc = data.frame(index= seq(1:4),fone=fone.val,recall=recall.val,algo= c("Linear SVM","Lin
boxplot_df95pca2opc= data.frame("Values"=c(tmp2recalinsvm,tmp2recalinsvmop,tmp2recaradc,tmp2recaradcop,

ggplot(data=boxplot_df95pca2opc, aes(x=Classifiers,y=Values, fill=Classifiers)) +   geom_boxplot(outlie
```

The below code is for line graph to compare classifiers with default and optimized parameters

```
ggplot(data=plot_df95pca2opc ,aes(x=index,y=fone))+geom_line(data=plot_df95pca2opc ,aes(x=index,y=fone,
```

The below code is for calculating ROC of the curve

```
library(caret)
set.seed(1)

#Split data into traning and validatio data set
inTrain <- createDataPartition(arrhythmia_labels, p = .6)[[1]]
train <- processed_arrhythmia_df_nolabels[ inTrain,-which(names(processed_arrhythmia_df_nolabels) %in% c
test  <- processed_arrhythmia_df_nolabels[-inTrain,-which(names(processed_arrhythmia_df_nolabels) %in% c
cls.train <- arrhythmia_labels[inTrain]
cls.test <- arrhythmia_labels[-inTrain]

#apply PCA on training dataset
arrhythmia.pca <- prcomp(train)
tot.var <- sum(arrhythmia.pca$sdev^2)
var.explained <- data.frame(pc = seq(1:ncol(processed_arrhythmia_df_nolabels[-fold[[i]],-which(names(pr

cum_var.explained =c()
for (u in seq(nrow(var.explained)))
{cum_var.explained =  c(cum_var.explained,sum(var.explained[1:u,2])*100)}

#Identif number of PCs to use to explain 95% variability
pctouse = min(which(cum_var.explained > 95))

oversampled_df = data.frame(arrhythmia.pca$x[,1:pctouse])
oversampled_df$class = cls.train
all_exis_classes  = sort(unique(arrhythmia_labels))
for (eachminroclass in all_exis_classes[!all_exis_classes %in% 1])
{
balanceddata<-ovun.sample(class ~ ., data =subset(oversampled_df,class == eachminroclass | class == 1),
oversampled_df = rbind(subset(balanceddata,class != 1),subset(oversampled_df,class != eachminroclass))
}
oversampled_df <- oversampled_df[sample(nrow(oversampled_df)),]  #Shuffling rows
arrhythmia.pcatest = predict(arrhythmia.pca, test)


radsvm.model1 <- svm(x = oversampled_df[,-which(colnames(oversampled_df) %in% c("class")),drop=FALSE], y
#Calculate predictions on validation data
preds <- predict(radsvm.model1, newdata=data.frame(arrhythmia.pcatest[,1:pctouse]))

#Multiclass ROC under the curve calculation
library(pROC)
y_pred<-as.ordered(preds)
auc <- multiclass.roc(cls.test, y_pred)
print(auc(auc))
```