Estatística e Análise de Dados

José Pinto, Nirbhaya Shaji

21/04/2020

Introduction

This work has been performed to develop experience in a full data science and analysis pipeline, obtain the data, visualize its features, pre-process it should it be necessary, model the data and make predictions and evaluate the results. An emphasis has been made in the modeling, with the models being restricted to the ones taught in class.

Goals

Classification and predictions on a real world dataset. Obtain practical and theoretical experience with classification tasks using a subset of models.

Models

In the work presented below we worked with several different models for classification. The models are linear regression (LR), multinomial logistic regression (MLR), linear discriminant analysis (LDA) and quadratic discriminant analysis (QDA)br

For LR and MLR we also use LASSO and RIDGE variants.

Two unsupervised clustering models were also employed, hierarchical clustering and expectation maximization for mixture distributions (EMMD). Both of these were not used for classification, but for further data analisys.

Dataset

The dataset we are going to work with is the 2019 "World Happiness Report" dataset available in kaggle "https://www.kaggle.com/unsdsn/world-happiness/data" The dataset contains information about country, region and several other variables used to calculate the happiness score.

The dataset is comprised of a total of 156 observations (rows/countries) and 9 variables (columns), 1 target and 8 predictors. The variables are as follows:

Target variable: Score - The happiness score obtained based on the other features (continuous value)

Predictor variables: Overall rank - Rank of the country based on the Happiness Score. (integer value) Country or region - Country the data belongs to (categorical value) GDP per capita - Gross domestic product per person (continuous value) Social support - Amount of social support received (continuous value) Healthy life expectancy - Average life expectancy of individuals (continuous value)) Freedom to make life choices - Amount of freedom to make choices (continuous value) Generosity - Amount of generosity (continuous value) Perceptions of corruption - Perceived amount of corruption in country (continuous value)

All the required libraries are imported here to more easily identify dependencies.

```
library(mlbench)
library(class)
library(GGally)
library(magrittr)
library(MASS)
```

```
library(hmeasure)
library(randomForest)
library(reshape2)
library(glmnet)
library(ggplot2)
library(reshape2)
library(nnet)
library(MLmetrics)
library(MASS)
library(tidyverse)
library(CrossValidate)
library(mclust)
library(klaR)
library(caret)
library(dendextend)
```

As the work is intended to be performed on a categorical target the variables will be discretized into intervals. Due to the nature of the data both "Overall rank" and "Country or region" will be removed.

Import the data and check variable types.

```
WHR = read.csv("2019.csv")
WHR = WHR[3:9]
str(WHR)
```

```
## 'data.frame':    156 obs. of  7 variables:
##  $ Score                       : num  7.77 7.6 7.55 7.49 7.49 ...
##  $ GDP.per.capita              : num  1.34 1.38 1.49 1.38 1.4 ...
##  $ Social.support              : num  1.59 1.57 1.58 1.62 1.52 ...
##  $ Healthy.life.expectancy     : num  0.986 0.996 1.028 1.026 0.999 ...
##  $ Freedom.to.make.life.choices: num  0.596 0.592 0.603 0.591 0.557 0.572 0.574 0.585 0.584 0.532 ..
##  $ Generosity                  : num  0.153 0.252 0.271 0.354 0.322 0.263 0.267 0.33 0.285 0.244 ...
##  $ Perceptions.of.corruption   : num  0.393 0.41 0.341 0.118 0.298 0.343 0.373 0.38 0.308 0.226 ...
```

First entries of the data.

```
head(WHR)
```

```
##   Score GDP.per.capita Social.support Healthy.life.expectancy
## 1 7.769          1.340          1.587                   0.986
## 2 7.600          1.383          1.573                   0.996
## 3 7.554          1.488          1.582                   1.028
## 4 7.494          1.380          1.624                   1.026
## 5 7.488          1.396          1.522                   0.999
## 6 7.480          1.452          1.526                   1.052
##   Freedom.to.make.life.choices Generosity Perceptions.of.corruption
## 1                        0.596      0.153                     0.393
## 2                        0.592      0.252                     0.410
## 3                        0.603      0.271                     0.341
## 4                        0.591      0.354                     0.118
## 5                        0.557      0.322                     0.298
## 6                        0.572      0.263                     0.343
```

Summary of data distribution.

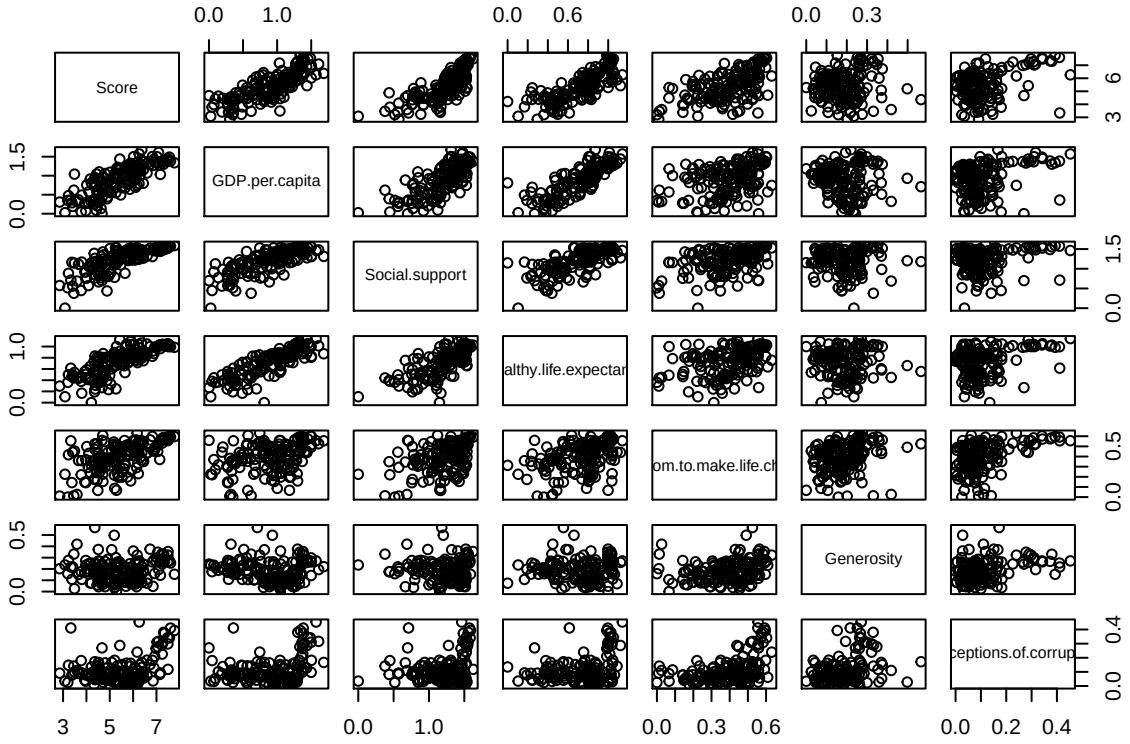
```
summary(WHR)
```

```
##      Score       GDP.per.capita   Social.support  Healthy.life.expectancy
```

```
##  Min.   :2.853   Min.   :0.0000   Min.   :0.000   Min.   :0.0000
##  1st Qu.:4.545   1st Qu.:0.6028   1st Qu.:1.056   1st Qu.:0.5477
##  Median :5.380   Median :0.9600   Median :1.272   Median :0.7890
##  Mean   :5.407   Mean   :0.9051   Mean   :1.209   Mean   :0.7252
##  3rd Qu.:6.184   3rd Qu.:1.2325   3rd Qu.:1.452   3rd Qu.:0.8818
##  Max.   :7.769   Max.   :1.6840   Max.   :1.624   Max.   :1.1410
##  Freedom.to.make.life.choices   Generosity     Perceptions.of.corruption
##  Min.   :0.0000               Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.3080               1st Qu.:0.1087   1st Qu.:0.0470
##  Median :0.4170               Median :0.1775   Median :0.0855
##  Mean   :0.3926               Mean   :0.1848   Mean   :0.1106
##  3rd Qu.:0.5072               3rd Qu.:0.2482   3rd Qu.:0.1412
##  Max.   :0.6310               Max.   :0.5660   Max.   :0.4530
```

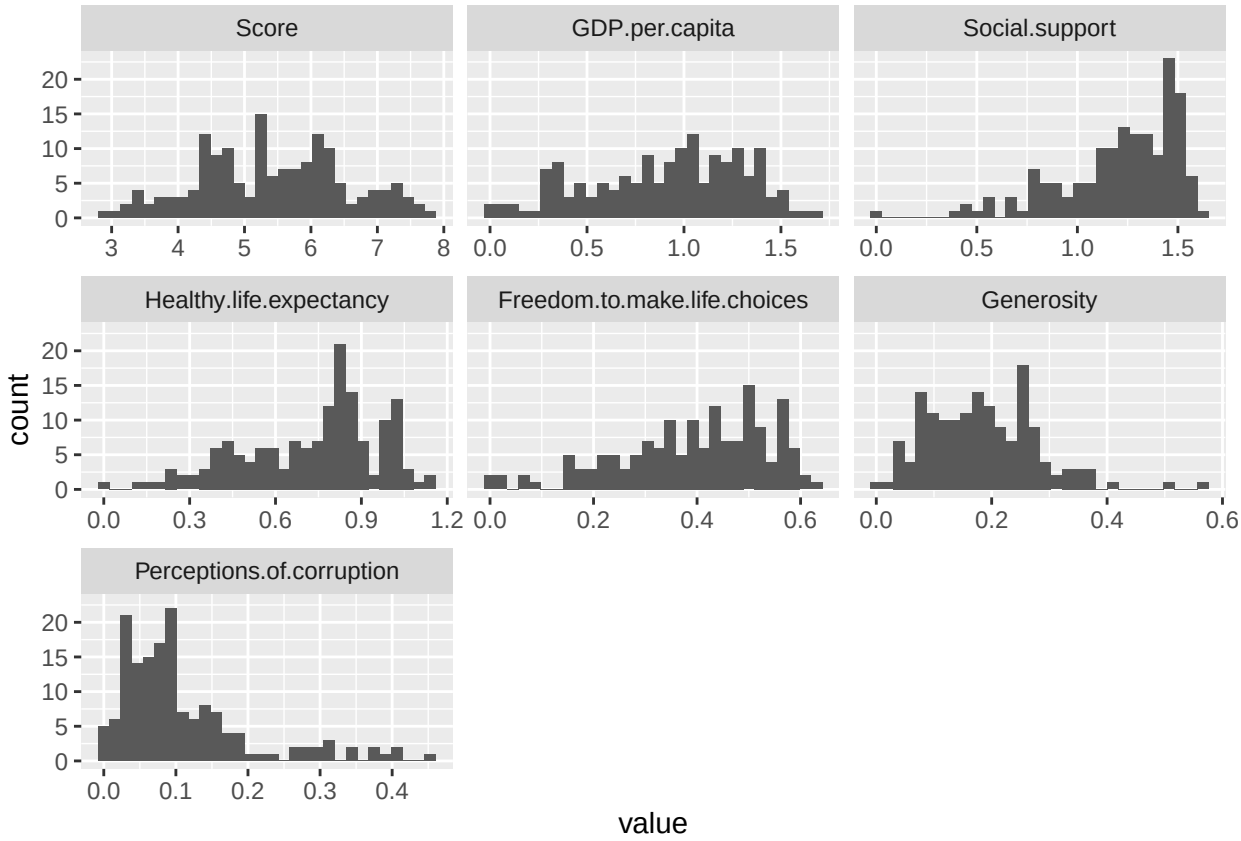
Scatterplots of data.

```
pairs(WHR)
```

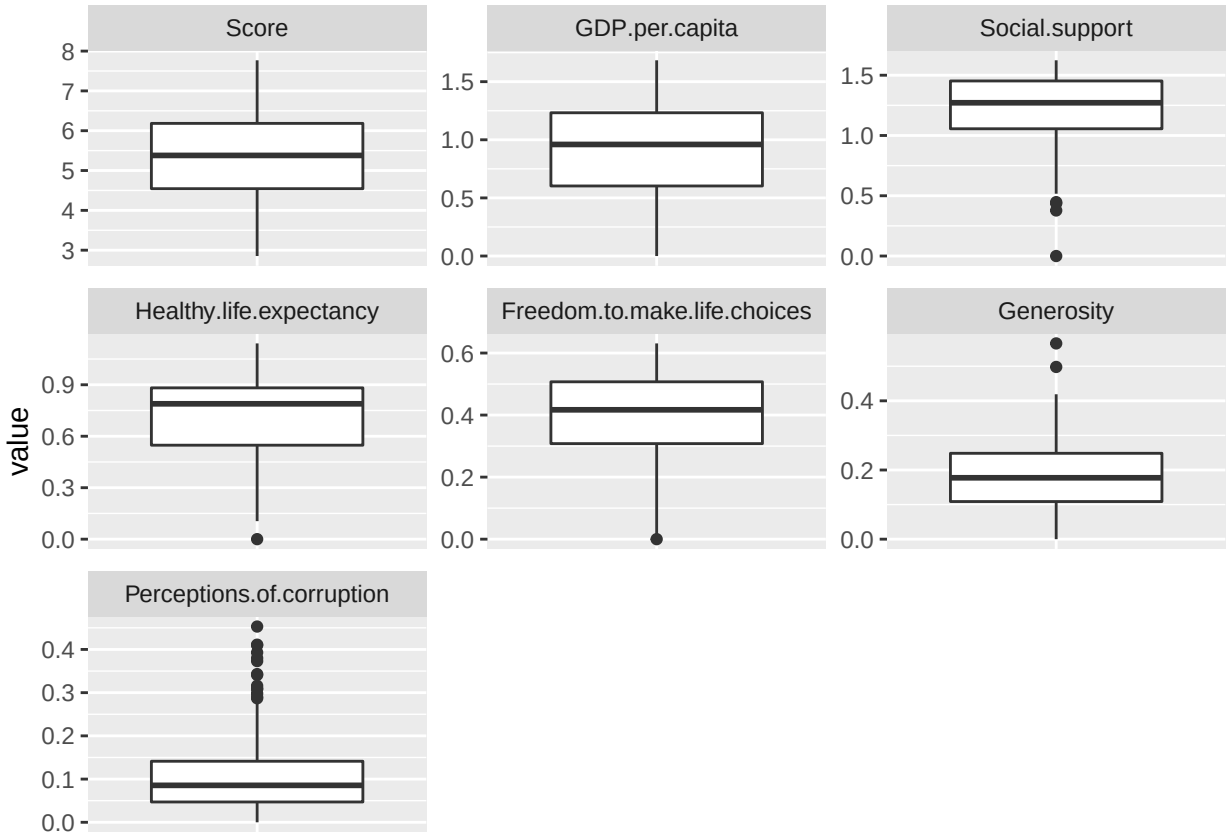


Histograms and boxplots of data.

```
data = melt(WHR)
ggplot(data,aes(x=value))+facet_wrap(~variable,scales = "free_x")+geom_histogram()
```




```
ggplot(data, aes(factor(variable), value)) +
  geom_boxplot() + facet_wrap(~variable, scale="free") +
  theme(axis.title.x=element_blank(),
        axis.text.x=element_blank(),
        axis.ticks.x=element_blank())
```




```
#set the seed so we can replicate results
set.seed(123)
```

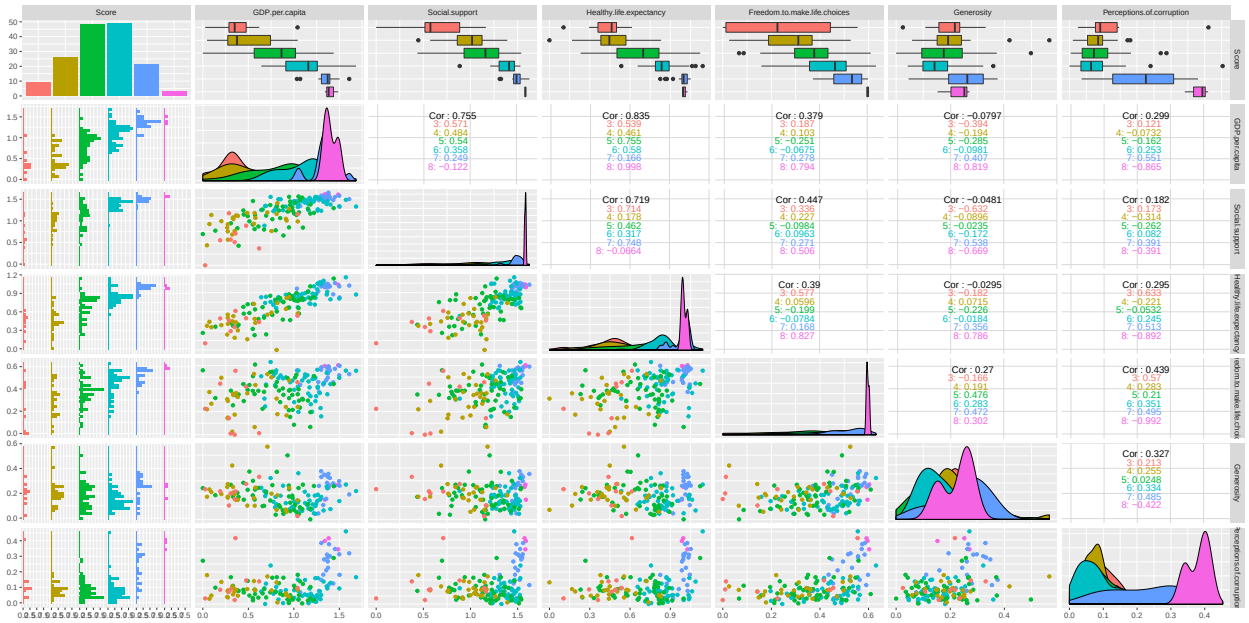
Discretize target variable

Due to the values that the target variable (score) takes, the simplest way to discretize it is just to round.

```
WHRDiscrete = WHR
WHRDiscrete$Score = as.factor(round(WHRDiscrete$Score))
```

Data distribution per target variable.

```
WHRDiscrete %>% ggpairs(.,mapping=ggplot2::aes(colour=Score));
```



Data Split

Now that we have the data we split it into train and test data.

```
targetColumn = 1

#set percentage of data for training to 80%
trainPercent = 0.8
#get the indices of the rows for training
trainRows = balancedSplit(WHRDiscrete$Score, trainPercent)
#get the rows with the training indices
trainWHR = WHRDiscrete[trainRows,-targetColumn]
trainWHRY = WHRDiscrete[trainRows,targetColumn]
#get the rows that are not training indices (test data)
testWHR = WHRDiscrete[not(trainRows),-targetColumn]
testWHRY = WHRDiscrete[not(trainRows),targetColumn]
```

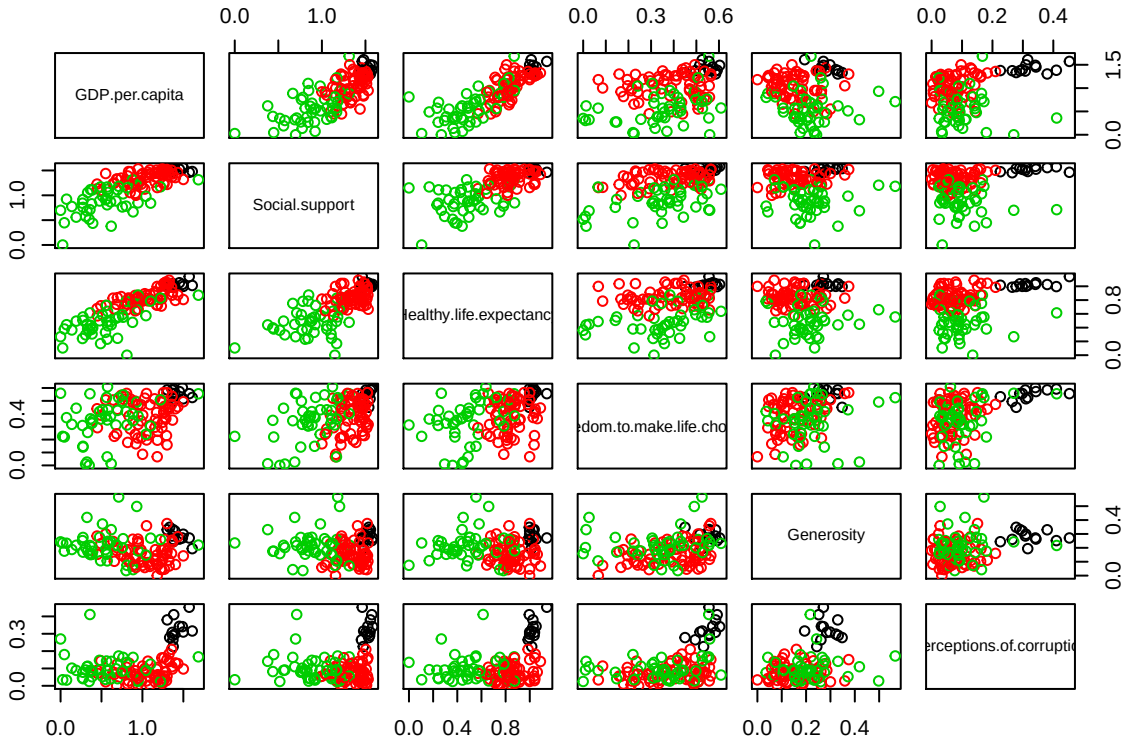
Modeling

In this section we train the models and check the learned parameters.

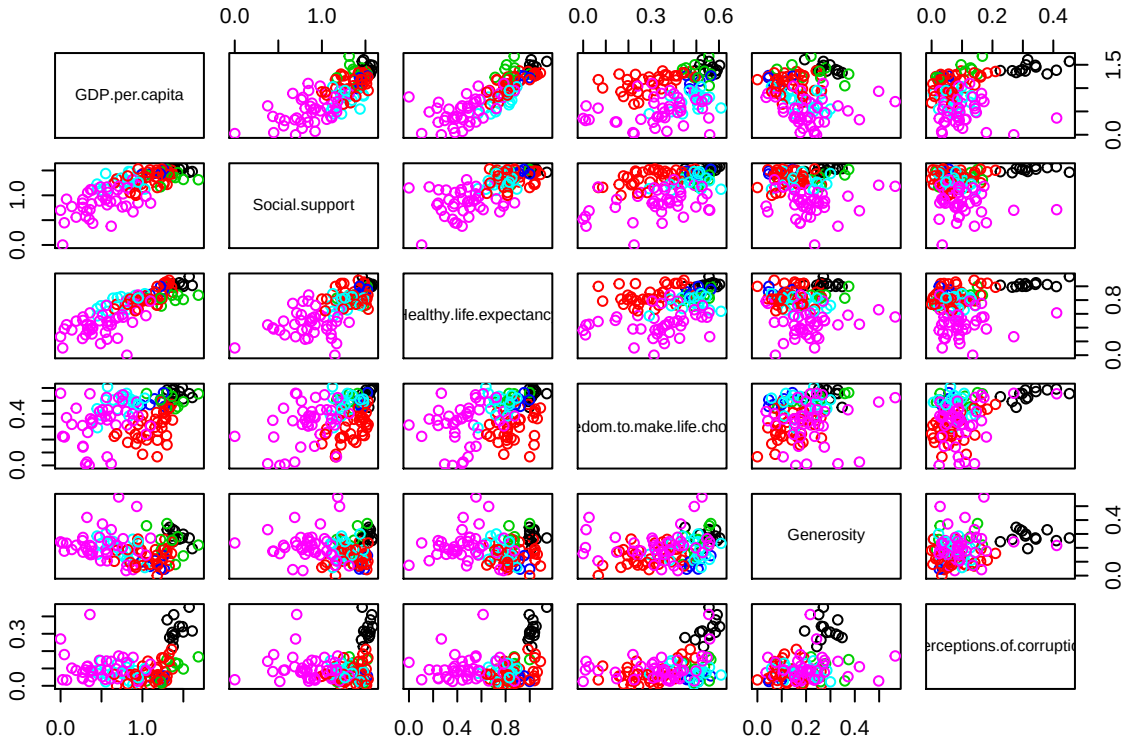
Expectation Maximization for Mixture Distributions - EMMD

We face a classification problem, and as this method is a clustering one, without any way to create reasonable predictions, we will use it only for further data analysis.

```
#create a EMMD model with automatically picked number of clusters
EMMDModel = Mclust(trainWHR)
#visualize clusters
pairs(trainWHR,col = EMMDModel$classification)
```




```
#create a EMMD model with the number of clusters being the number of classes
EMMDFixedModel = Mclust(trainWHR,G=6)
#visualize clusters
pairs(trainWHR,col = EMMDFixedModel$classification)
```



The leaned model without restrictions selected 3 clusters, which is not the number of classes in our data. This however is to be expected, given that the data was split into mostly arbitrary classes. This division could indicate groups of countries with similar and distinct characteristics. This is slightly amusing, given the colloquial description of countries as being first world, second world and third world. When we force the data into our 6 classes, the results are as expected poor, with the number in each class being radically different from the real numbers.

Borders

We will for all the models present the decision boundaries. Below is the function used to plot them. Due to file size restrictions (to deliver by mail), only one border will be shown per model. To change this comment the "return()" line in each model border drawing loop.

```
#draw the decision boundaries for the model
boundaryPlot = function(model, X, Y, predFunction, resolution = 150, newData = NULL, newDataY = NULL, pa
  if(is.null(newData)){
    newData = X
    newDataY = Y
  }

  #get ranes for the data (min and max)
  ranges = sapply(newData[,c(1:2)], range)
  #get n points bettwen the ranges
  xAxis = seq(ranges[1,1], ranges[2,1], length.out = resolution)
  yAxis = seq(ranges[1,2], ranges[2,2], length.out = resolution)
  #create a grid of points
```

```
  grid = data.frame(expand.grid(xAxis,yAxis))

  names(grid) = names(X)
  pred = predFunction(model, grid, newDataY, parameters)

  #get the values for contour
  numericPred = as.numeric(pred)-1

  plotz=ggplot(data=grid, aes(grid[,1], grid[,2])) +
    #place background color for predicions
    geom_point(aes(colour=as.factor(pred)), size=4,shape=16,alpha=.02) +
    #place prediction borders
    geom_contour(aes(z=numericPred), colour='black', size = 2,alpha=1) +
    #place data points
    geom_point(data=newData, aes(newData[,1], newData[,2], colour=as.factor(newDataY)),size=3,shape=16,a
    #select display colors
    scale_color_manual(values=c("#0000FF", "#00FF00", "#00FFFF", "#FF0000", "#FF00FF", "#FFFF00")) +
    theme_bw() +
    #set x and y labels
    xlab(names(X)[1]) +
    ylab(names(X)[2])

  return(plotz)
}
```

LR

We will use linear regression to predict the classes. Linear regression is typically not used for classification tasks, especially for multiple classes. In order to perform this we will create dummy variables and train the data to predict the values of each dummy. Then select the target variable with the highest predicted value. We expect really bad results.

```
#turn taget variable into set of dummy variables
dummyTargets = model.matrix(~trainWHRY+0)
#train model
LRModel = lm(dummyTargets~.,data = trainWHR)
```

LR Borders

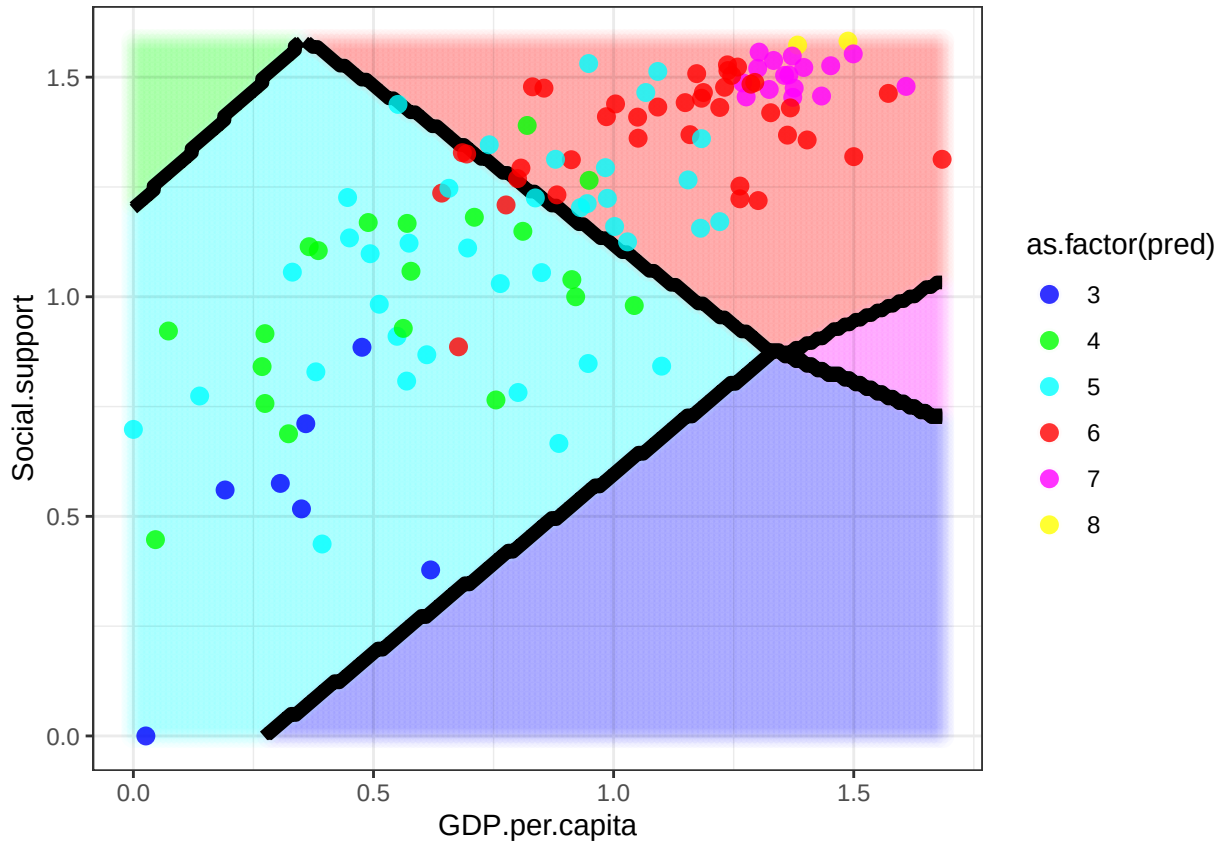
Here we present the decision boundaries using subsets of two variables.

```
#function to obtain LR predictions from X
LRPredFunction = function(model, X, Y, ...){
  #get the predicted values for the grid
  rawPred = predict(model, X)
  pred=c()
  for (row in 1:length(rawPred[,1])) {
    maxProb = max(rawPred[row,])
    maxIndex = match(maxProb,rawPred[row,])
    maxValue = levels(Y)[maxIndex]
    pred = c(pred,maxValue)
  }
  return(pred)
```

```
}

#get the names of the columns in the final model
columns = names(trainWHR)
#get number of columns
ncols = length(columns)

#select the first column
for(column1 in seq(1,ncols-1,1)){
  #select a diferent second column that has not been a first columns
  for(column2 in seq(column1+1,ncols,1)){
    WHRToBorder = trainWHR[,c(columns[column1],columns[column2])]
    LRBorderModel = lm(dummyTargets~.,data = WHRToBorder)
    print(boundaryPlot(LRBorderModel,WHRToBorder,trainWHRY,LRPredFunction))
    #comment next line to show all borders
    return()
  }
}
```



LR Lasso

We will use LASSO linear regression to predict the classes. LASSO Linear regression is typically not used for classification tasks, especially for multiple classes. In order to perform this we will create dummy variables and train the data to predict the values of each dummy. Then select the target variable with the highest predicted value. We expect really bad results.

```
#turn taget variable into set of dummy variables
dummyTargets = model.matrix(~trainWHRY+0)
#find the best lambda value
LRLassoLambda = cv.glmnet(as.matrix(trainWHR), dummyTargets, alpha = 1, family = "mgauss")$lambda.1se

#train model
LRLassoModel = glmnet(as.matrix(trainWHR), dummyTargets, alpha = 1, lambda = LRLassoLambda, family = "m
```

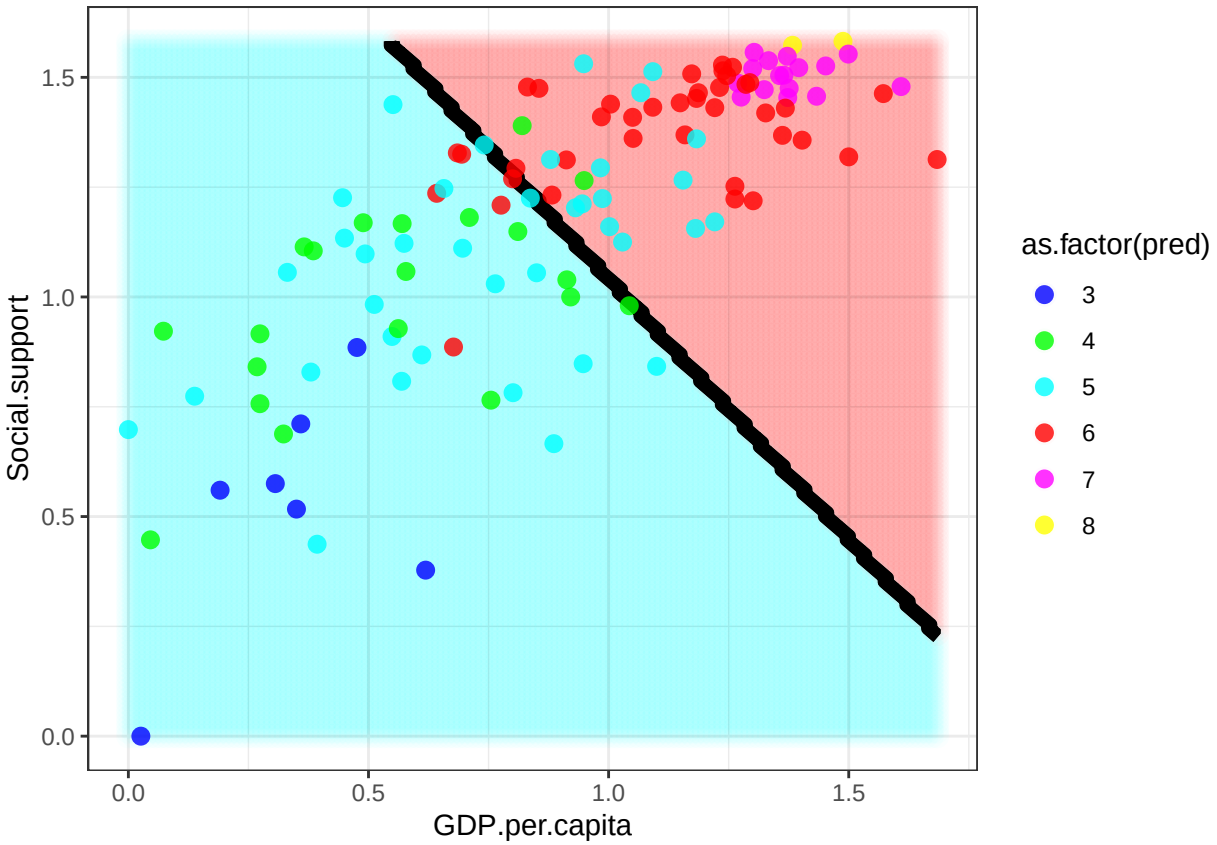
LR Lasso Borders

Here we present the decision boundaries using subsets of two variables.

```
#function to obtain LR Lasso predictions from X
LRLassoPredFunction = function(model, X, Y, parameters){
  rawPred = predict(model,s=parameters[["lambda"]], as.matrix(X))[,,1]
  pred=c()
  for (row in 1:length(rawPred[,1])) {
    maxProb = max(rawPred[row,])
    maxIndex = match(maxProb,rawPred[row,])
    maxValue = levels(Y)[maxIndex]
    pred = c(pred,maxValue)
  }
  return(pred)
}

#get the names of the columns in the final model
columns = names(trainWHR)
#get number of columns
ncols = length(columns)

#select the first column
for(column1 in seq(1,ncols-1,1)){
  #select a diferent second column that has not been a first columns
  for(column2 in seq(column1+1,ncols,1)){
    WHRToBorder = trainWHR[,c(columns[column1],columns[column2])]
    lambda = cv.glmnet(as.matrix(WHRToBorder), dummyTargets, alpha = 1, family = "mgauss")$lambda.1se
    LRLassoBorderModel = glmnet(as.matrix(WHRToBorder), dummyTargets, alpha = 1, lambda = lambda, family
    print(boundaryPlot(LRLassoBorderModel,WHRToBorder,trainWHRY,LRLassoPredFunction, parameters = list(
    #comment next line to show all borders
    return()
  }
}
```



LR Ridge

We will use Ridge linear regression to predict the classes. Ridge Linear regression is typically not used for classification tasks, especially for multiple classes. In order to perform this we will create dummy variables and train the data to predict the values of each dummy. Then select the target variable with the highest predicted value. We expect really bad results.

```
#turn taget variable into set of dummy variables
dummyTargets = model.matrix(~trainWHRY+0)
#find the best lambda value
LRRidgeLambda = cv.glmnet(as.matrix(trainWHR), dummyTargets, alpha = 0, family = "mgauss")$lambda.1se

#train model
LRRidgeModel = glmnet(as.matrix(trainWHR), dummyTargets, alpha = 0, lambda = LRRidgeLambda, family = "m
```

LR Ridge Borders

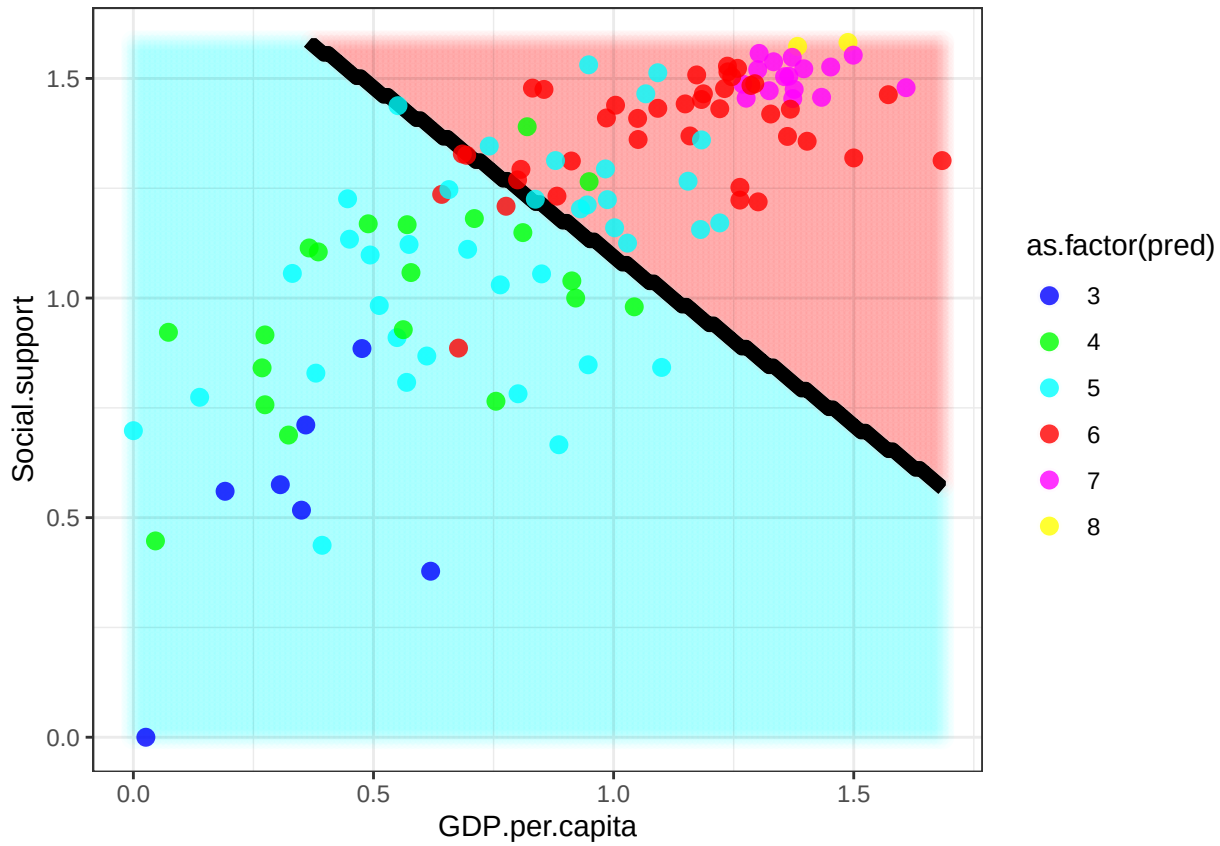
Here we present the decision boundaries using subsets of two variables.

```
#function to obtain LR Ridge predictions from X
LRRidgePredFunction = function(model, X, Y, parameters){
  rawPred = predict(model,s=parameters[["lambda"]], as.matrix(X))[,,1]
  pred=c()
  for (row in 1:length(rawPred[,1])) {
    maxProb = max(rawPred[row,])
```

```
    maxIndex = match(maxProb,rawPred[row,])
    maxValue = levels(Y)[maxIndex]
    pred = c(pred,maxValue)
  }
  return(pred)
}

#get the names of the columns in the final model
columns = names(trainWHR)
#get number of columns
ncols = length(columns)

#select the first column
for(column1 in seq(1,ncols-1,1)){
  #select a diferent second column that has not been a first columns
  for(column2 in seq(column1+1,ncols,1)){
    WHRToBorder = trainWHR[,c(columns[column1],columns[column2])]
    lambda = cv.glmnet(as.matrix(WHRToBorder), dummyTargets, alpha = 0, family = "mgauss")$lambda.1se
    LRRidgeBorderModel = glmnet(as.matrix(WHRToBorder), dummyTargets, alpha = 0, lambda = lambda, family
    print(boundaryPlot(LRRidgeBorderModel,WHRToBorder,trainWHRY,LRRidgePredFunction, parameters = list(
    #comment next line to show all borders
    return()
  }
}
```



MLR

The MLR (multinomial logistic regression) is the generalization of logistic regression for more than 2 classes. The library used utilizes neural networks for a more efficient approximation. Due to it being an approximation no significance values are available for variable selection. The AIC values after training will be used for stepwise variable selection.

```
#perform backwards stepwise parameter search on multinomial model
multinomialStepWise = function(data,dataY){
  minMLRModel = multinom(dataY~., data=data, trace=FALSE)
  minData = data
  minAic = minMLRModel$AIC
  while(TRUE){
    prevAic = minAic
    for (column in names(data)) {
      #remove one column from the data
      reducedData = data
      reducedData[column] = NULL

      #get a new model with the reduced data
      MLRModel = multinom(dataY~., data=reducedData)

      #check if aic decreased
      if(MLRModel$AIC < minAic){
        #update the best model found
        minAic = MLRModel$AIC
        minMLRModel = MLRModel
        minData = reducedData
      }
    }
    #if model didnt change return it
    if(prevAic == minAic){
      return(minMLRModel)
    }
    #restrict the data for the next loop
    data=minData
  }
}

MLRModel = multinomialStepWise(trainWHR,trainWHRY)
```

```
MLRModel
```

```
## Call:
## multinom(formula = dataY ~ ., data = reducedData)
##
## Coefficients:
##   (Intercept) Social.support Healthy.life.expectancy
## 4   -3.528979       8.195561               -2.323994
## 5   -6.266649       8.276838                1.346981
## 6  -23.345857      15.457369                8.631232
## 7  -69.268975      37.496984               19.274329
## 8 -134.211073      69.964498               -2.477817
##   Freedom.to.make.life.choices
## 4                   -2.137961
```

```
## 5                     1.435637
## 6                     7.116487
## 7                    13.209367
## 8                    74.378658
##
## Residual Deviance: 186.1466
## AIC: 226.1466
```

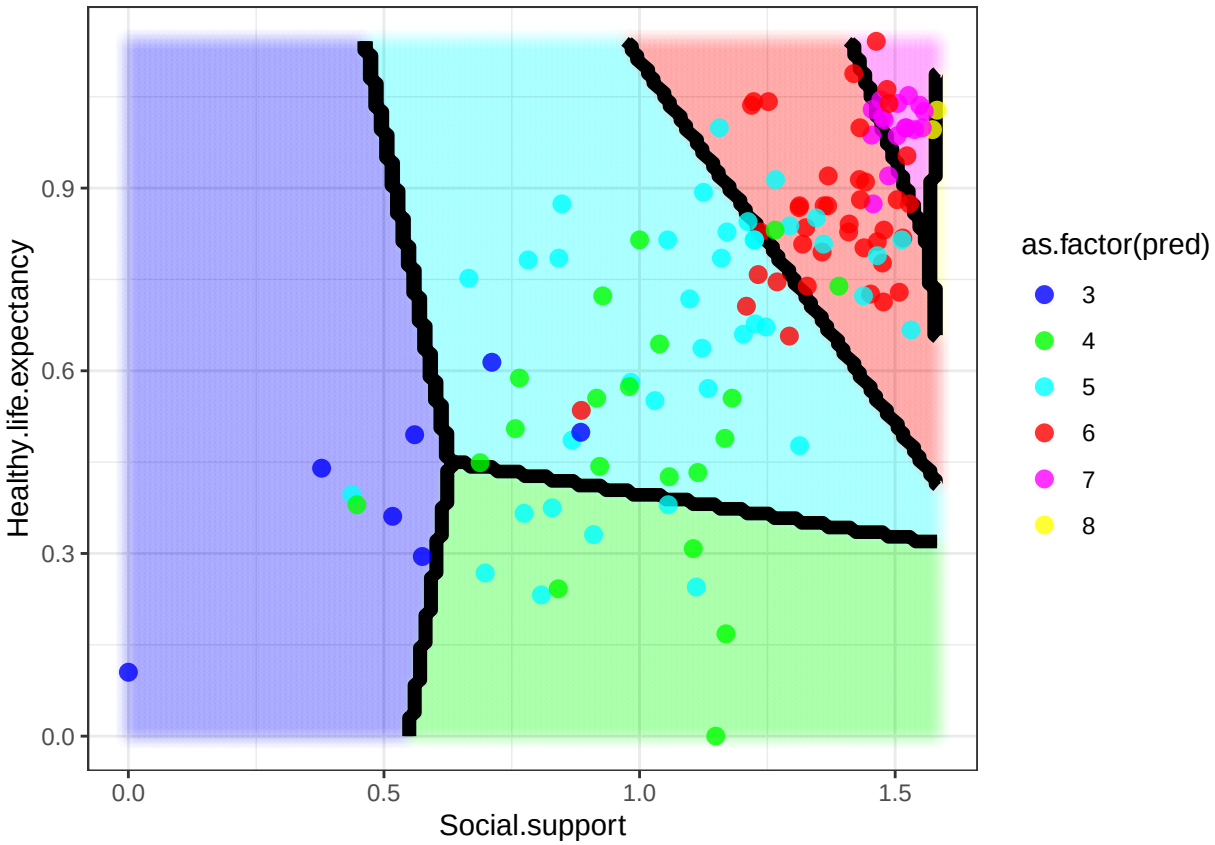
MLR Borders

Here we present the decision boundaries using subsets of two variables. We will only present for the variables present in the final model.

```
#function to obtain MLR predictions from X
MLRPredFunction = function(model, X, ...){
  return(predict(model, X))
}

#get the names of the columns in the final model
columns = MLRModel$coefnames[-1]
#get number of columns
ncols = length(columns)

#select the first column
for(column1 in seq(1,ncols-1,1)){
  #select a diferent second column that has not been a first columns
  for(column2 in seq(column1+1,ncols,1)){
    WHRToBorder = trainWHR[,c(columns[column1],columns[column2])]
    MLRBorderModel = multinom(trainWHRY~., data=WHRToBorder)
    print(boundaryPlot(MLRBorderModel,WHRToBorder,trainWHRY,MLRPredFunction))
    #comment next line to show all borders
    return()
  }
}
```



MLR Lasso

We will use LASSO multinomial logistic regression to predict the classes. In order to perform this we will create dummy variables and train the data to predict the values of each dummy. Then select the target variable with the highest predicted value.

```
#turn taget variable into set of dummy variables
dummyTargets = model.matrix(~trainWHRY+0)

success = FALSE
#due to randomness mlr sometimes gives an error. run until success
while(not(success)){
  try(
    {
      #find the best lambda value
      MLRLassoLambda = cv.glmnet(as.matrix(trainWHR), dummyTargets, alpha = 1, family = "multinomial")$
      #train model
      MLRLassoModel = glmnet(as.matrix(trainWHR), dummyTargets, alpha = 1, lambda = MLRLassoLambda, fami
      success = TRUE
    }
  )
}
```

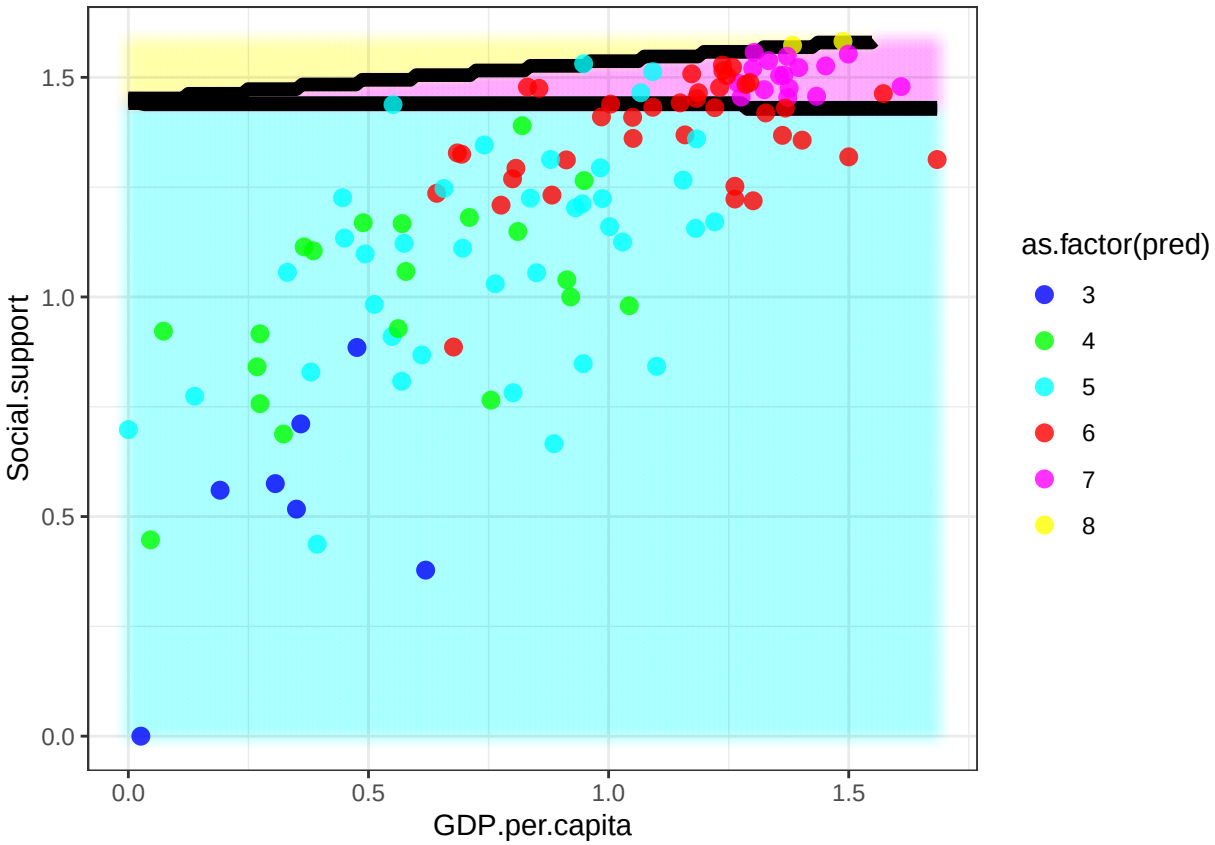
MLR Lasso Borders

Here we present the decision boundaries using subsets of two variables.

```
#function to obtain MLR Lasso predictions from X
MLRLassoPredFunction = function(model, X, Y, parameters){
  rawPred = predict(model,s=parameters[["lambda"]], as.matrix(X))[,,1]
  pred=c()
  for (row in 1:length(rawPred[,1])) {
    maxProb = max(rawPred[row,])
    maxIndex = match(maxProb,rawPred[row,])
    maxValue = levels(Y)[maxIndex]
    pred = c(pred,maxValue)
  }
  return(pred)
}

#get the names of the columns in the final model
columns = names(trainWHR)
#get number of columns
ncols = length(columns)

#select the first column
for(column1 in seq(1,ncols-1,1)){
  #select a diferent second column that has not been a first columns
  for(column2 in seq(column1+1,ncols,1)){
    WHRToBorder = trainWHR[,c(columns[column1],columns[column2])]
    success = FALSE
    #due to randomness mlr sometimes gives an error. run until success
    while(not(success)){
      try(
        {
          #find the best lambda value
          lambda = cv.glmnet(as.matrix(WHRToBorder), dummyTargets, alpha = 1, family = "multinomial")$la
          #train model
          MLRLassoBorderModel = glmnet(as.matrix(WHRToBorder), dummyTargets, alpha = 1, lambda = lambda
          success = TRUE
        }
      )
    }
    print(boundaryPlot(MLRLassoBorderModel,WHRToBorder,trainWHRY,MLRLassoPredFunction,parameters = list
    #comment next line to show all borders
    return()
  }
}
```



MLR Ridge

We will use RIDGE multinomial logistic regression to predict the classes. In order to perform this we will create dummy variables and train the data to predict the values of each dummy. Then select the target variable with the highest predicted value.

```
#turn taget variable into set of dummy variables
dummyTargets = model.matrix(~trainWHRY+0)

success = FALSE
#due to randomness mlr sometimes gives an error. run until success
while(not(success)){
  try(
    {
      #find the best lambda value
      MLRRidgeLambda = cv.glmnet(as.matrix(trainWHR), dummyTargets, alpha = 0, family = "multinomial")$
      #train model
      MLRRidgeModel = glmnet(as.matrix(trainWHR), dummyTargets, alpha = 0, lambda = MLRRidgeLambda, fami
      success = TRUE
    }
  )
}
```

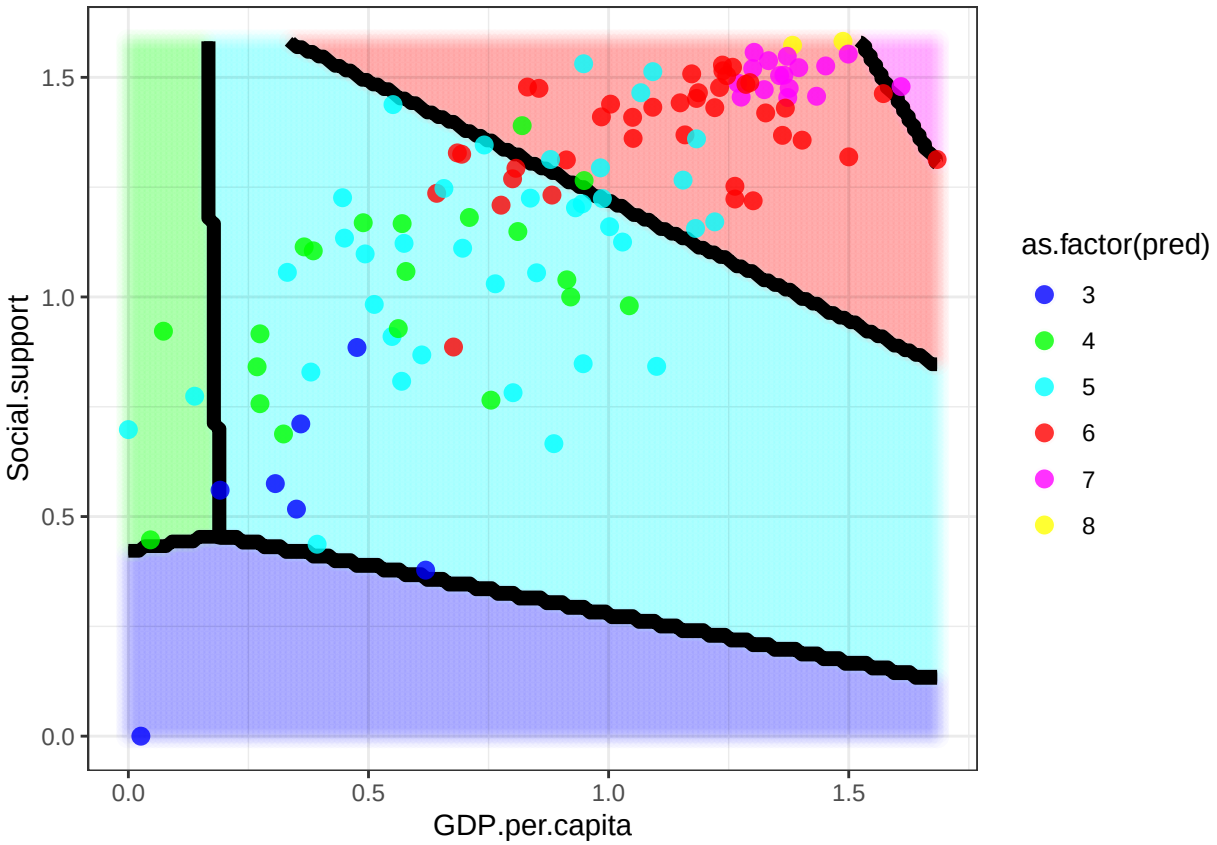
MLR Ridge Borders

Here we present the decision boundaries using subsets of two variables.

```
#function to obtain MLR Ridge predictions from X
MLRRidgePredFunction = function(model, X, Y, parameters){
  rawPred = predict(model,s=parameters[["lambda"]], as.matrix(X))[,,1]
  pred=c()
  for (row in 1:length(rawPred[,1])) {
    maxProb = max(rawPred[row,])
    maxIndex = match(maxProb,rawPred[row,])
    maxValue = levels(Y)[maxIndex]
    pred = c(pred,maxValue)
  }
  return(pred)
}

#get the names of the columns in the final model
columns = names(trainWHR)
#get number of columns
ncols = length(columns)

#select the first column
for(column1 in seq(1,ncols-1,1)){
  #select a diferent second column that has not been a first columns
  for(column2 in seq(column1+1,ncols,1)){
    WHRToBorder = trainWHR[,c(columns[column1],columns[column2])]
    success = FALSE
    #due to randomness mlr sometimes gives an error. run until success
    while(not(success)){
      try(
        {
          #find the best lambda value
          lambda = cv.glmnet(as.matrix(WHRToBorder), dummyTargets, alpha = 0, family = "multinomial")$la
          #train model
          MLRRidgeBorderModel = glmnet(as.matrix(WHRToBorder), dummyTargets, alpha = 0, lambda = lambda
          success = TRUE
        }
      )
    }
    print(boundaryPlot(MLRRidgeBorderModel,WHRToBorder,trainWHRY,MLRRidgePredFunction,parameters = list
    #comment next line to show all borders
    return()
  }
}
```



LDA

Here we use LDA as a classifier which is modeled using MASS R library. We get a couple of model parameters such as prior probabilities of groups, the group means and the coefficients of linear discriminant. The most important result here is the coefficients, they are values that describe the new feature space where the data will be project in. LDA reduces dimensionality from original number of feature to C — 1 features, where C is the number of classes. In this case, we have 6 classes, therefore the new feature space will have only 5 features.

```
table(WHRDiscrete$Score)
```

```
##
##  3  4  5  6  7  8
##  9 26 48 49 21  3
```

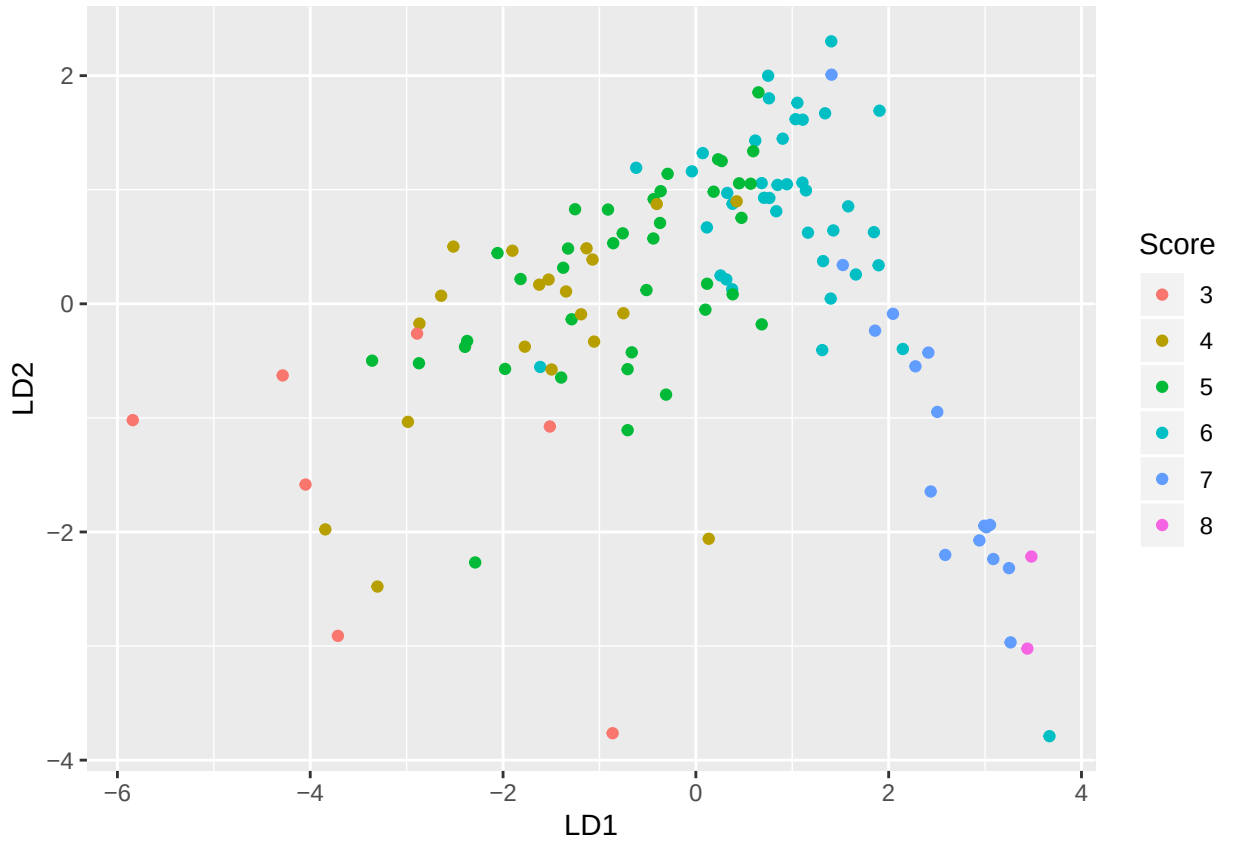
```
model_lda = lda(trainWHRY~., data = trainWHR)
model_lda
```

```
## Call:
## lda(trainWHRY ~ ., data = trainWHR)
##
## Prior probabilities of groups:
##          3          4          5          6          7          8
## 0.05737705 0.16393443 0.31147541 0.31967213 0.13114754 0.01639344
##
## Group means:
##   GDP.per.capita Social.support Healthy.life.expectancy
```

```
## 3      0.3324286       0.518000               0.4012857
## 4      0.5565000       0.994050               0.4933500
## 5      0.7599737       1.092789               0.6583421
## 6      1.1191795       1.375462               0.8601795
## 7      1.3772500       1.503250               1.0009375
## 8      1.4355000       1.577500               1.0120000
##   Freedom.to.make.life.choices Generosity Perceptions.of.corruption
## 3                    0.2375714  0.2338571                0.13414286
## 4                    0.2959500  0.2004000                0.08495000
## 5                    0.3593421  0.1788421                0.07705263
## 6                    0.4280000  0.1566154                0.08233333
## 7                    0.5080625  0.2568750                0.23775000
## 8                    0.5975000  0.2615000                0.37550000
##
## Coefficients of linear discriminants:
##                                     LD1         LD2       LD3         LD4
## GDP.per.capita                1.3412635  -0.2942896  1.401702 -1.51498188
## Social.support                2.6211581   0.9301375 -5.184554  0.06398950
## Healthy.life.expectancy       1.4798962   0.7156714  4.058673 -0.07619818
## Freedom.to.make.life.choices 2.4317235   2.7773992  2.644805  3.88141309
## Generosity                    0.4765424  -4.2086596 -1.342221 -9.60192085
## Perceptions.of.corruption     2.9789609 -12.4047164 -2.962681  5.58748551
##                                      LD5
## GDP.per.capita                 3.5738960
## Social.support                -0.6996928
## Healthy.life.expectancy       -4.0790958
## Freedom.to.make.life.choices -3.1020556
## Generosity                    -4.1107616
## Perceptions.of.corruption      1.9755673
##
## Proportion of trace:
##    LD1    LD2    LD3    LD4    LD5
## 0.7615 0.2006 0.0318 0.0047 0.0014
```

The below picture is the plot of the new feature space with the only two features, we can see the new position of the data, there is are points overlapping between the six classes, but in general, the dataset is pretty separable.

```
#LDA plot
lda.data = cbind(trainWHR, predict(model_lda)$x)
Score = trainWHRY
ggplot(lda.data, aes(LD1, LD2)) +
  geom_point(aes(color = Score))
```



QDA

QDA is implemented in R using the qda() function, which is also part of the MASS library. The syntax is identical to that of lda(). For qda() to work its necessary for each of the class to contain more number of instances than the number of predictors in the model. Since our data set ddnt have enough instances from the class 8 we combined class 8 and class 7 instances to try QDA on our data.

```
qda_trainWHRY = trainWHRY
qda_trainWHRY[qda_trainWHRY==8] = 7
qda_trainWHRY = droplevels(qda_trainWHRY)

model_qda <- qda(qda_trainWHRY~., data = trainWHR)
model_qda
```

```
## Call:
## qda(qda_trainWHRY ~ ., data = trainWHR)
##
## Prior probabilities of groups:
##         3         4         5         6         7
## 0.05737705 0.16393443 0.31147541 0.31967213 0.14754098
##
## Group means:
##   GDP.per.capita Social.support Healthy.life.expectancy
## 3      0.3324286       0.518000               0.4012857
## 4      0.5565000       0.994050               0.4933500
## 5      0.7599737       1.092789               0.6583421
```

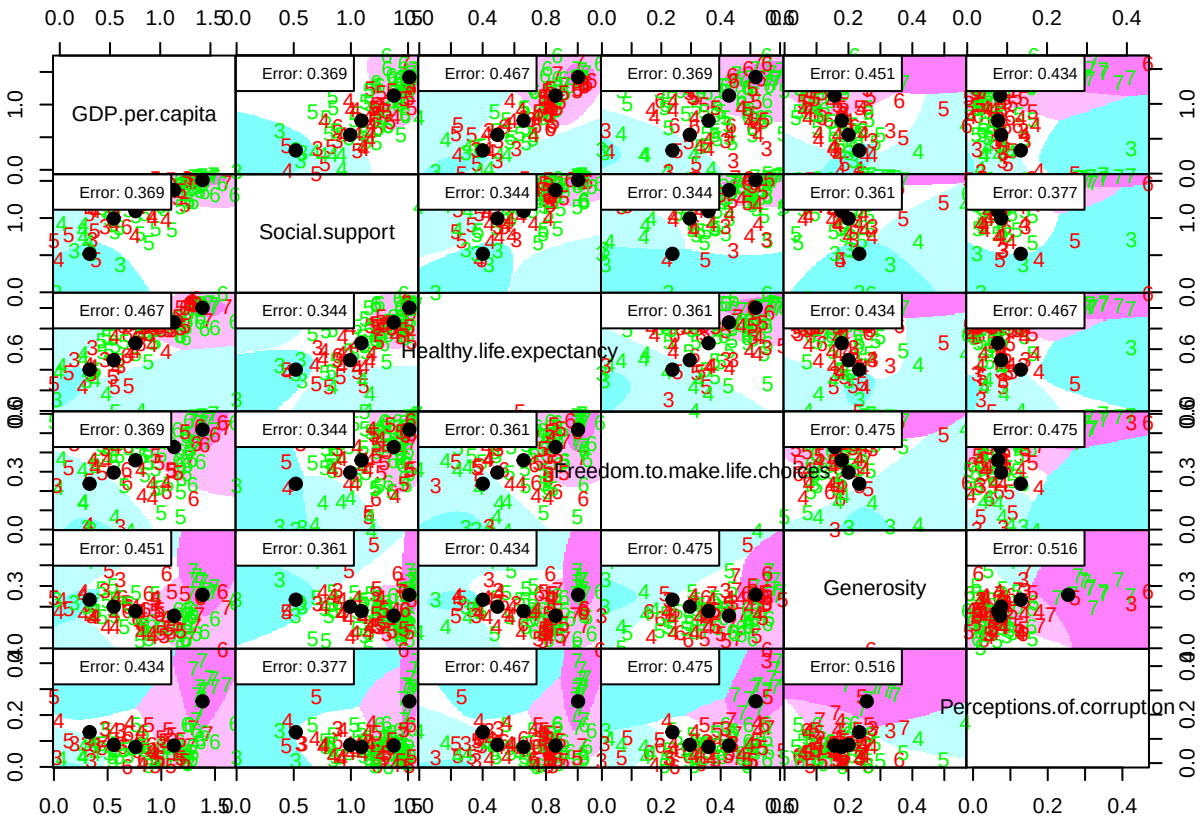
```
## 6      1.1191795       1.375462                0.8601795
## 7      1.3837222       1.511500                1.0021667
##   Freedom.to.make.life.choices Generosity Perceptions.of.corruption
## 3                    0.2375714  0.2338571                0.13414286
## 4                    0.2959500  0.2004000                0.08495000
## 5                    0.3593421  0.1788421                0.07705263
## 6                    0.4280000  0.1566154                0.08233333
## 7                    0.5180000  0.2573889                0.25305556
```

The output contains the group means. But it does not contain the coefficients of the linear discriminants, because the QDA classifier involves a quadratic, rather than a linear, function of the predictors.

The predict() function works in exactly the same fashion as for LDA.

The partimat( ) function in the klaR package can display the results of a quadratic classifications, 2 variables at a time. It provides a multiple figure array which shows the classification of observations for every combination of two variables. Moreover, the classification borders are displayed and the apparent error rates are given in each title.

```
partimat(qda_trainWHRY ~ ., data = trainWHR, method = "qda", plot.matrix = TRUE, col.correct='green', c
```

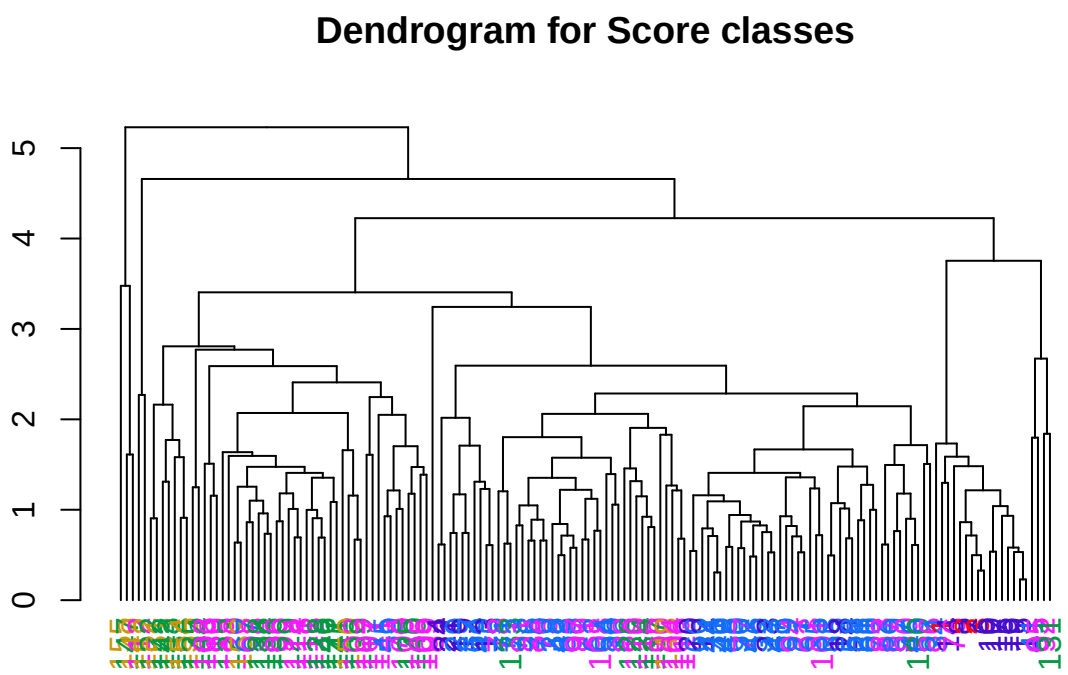


Hierarchical clustering

Using hclust() with the distance matrix we can use various techniques of cluster analysis for relationship discovery and plot a dendrogram that displays a hierarchical relationship among the classes.

```
WHRStd = as.data.frame(scale(WHR[2:7]))
```

```
clusters <- hclust(dist(WHRStd),method = 'average')
dend <- as.dendrogram(clusters)
COLS = c("#CC9709","#009942","#F818FF","#136CFF","#460CCC","#FF0012")
names(COLS) = unique(WHRDiscrete$Score)
dend <- color_labels(dend, col = COLS[WHRDiscrete$Score[labels(dend)]])
plot(dend, main = "Dendrogram for Score classes")
```



We can see from the above figure that the best choices for total number of clusters are either 4 or 3. Also averge linkage showed the best result amoung the available options in the hclust() function.

```
clusterCut <- cutree(clusters, 4)
table(clusterCut,WHRDiscrete$Score)
```

```
##
## clusterCut  3  4  5  6  7  8
##          1  0  1  3  2 11  3
##          2  6 24 44 47 10  0
##          3  1  0  1  0  0  0
##          4  2  1  0  0  0  0
```

```
clusterCut <- cutree(clusters, 3)
table(clusterCut,WHRDiscrete$Score)
```

```
##
## clusterCut  3  4  5  6  7  8
##          1  6 25 47 49 21  3
##          2  1  0  1  0  0  0
##          3  2  1  0  0  0  0
```

For the '3' cluster cut it looks like the algorithm successfully classified all the scores greater than 5. And for scores less than and equal to 5 the cluster cut '3' splited better than cluster '4'. Both the clusters had trouble with score class 3.

Predictions

In this section we make predictions using the previously trained models and evaluate how good they are.

LR

```
LRPred = LRPredFunction(LRModel,testWHR,testWHRY)
```

Accuracy.

```
Accuracy(LRPred,testWHRY)
```

```
## [1] 0.3823529
```

Confusion Matrix.

```
table(LRPred,testWHRY)
```

```
##       testWHRY
## LRPred 3 4 5 6 7 8
##      4 1 2 1 0 0 0
##      5 0 4 1 0 0 0
##      6 1 0 6 9 4 0
##      7 0 0 2 1 1 1
```

LR Lasso

```
LRLassoPred = LRLassoPredFunction(LRLassoModel,testWHR,testWHRY,parameters = list("lambda"=LRLassoLambda
```

Accuracy.

```
Accuracy(LRLassoPred,testWHRY)
```

```
## [1] 0.4117647
```

Confusion Matrix.

```
table(LRLassoPred,testWHRY)
```

```
##            testWHRY
## LRLassoPred  3  4  5  6  7  8
##           5  2  6  3  0  0  0
##           6  0  0  7 10  4  0
##           7  0  0  0  0  1  1
```

LR Ridge

```
LRRidgePred = LRRidgePredFunction(LRRidgeModel,testWHR,testWHRY,parameters = list("lambda"=LRRidgeLambda
```

Accuracy.

```
Accuracy(LRRidgePred,testWHRY)
```

```
## [1] 0.4117647
```

Confusion Matrix.

```
table(LRRidgePred,testWHRY)
```

```
##            testWHRY
## LRRidgePred  3  4  5  6  7  8
##           5  1  6  3  0  0  0
##           6  1  0  7 10  4  0
##           7  0  0  0  0  1  1
```

MLR

```
MLRPred = MLRPredFunction(MLRModel,testWHR,testWHRY)
```

Accuracy.

```
Accuracy(MLRPred,testWHRY)
```

```
## [1] 0.5588235
```

Confusion Matrix.

```
table(MLRPred,testWHRY)
```

```
##        testWHRY
## MLRPred 3 4 5 6 7 8
##       3 0 1 0 0 0 0
##       4 1 2 0 0 0 0
##       5 1 3 7 1 0 0
##       6 0 0 3 8 3 0
##       7 0 0 0 0 1 0
##       8 0 0 0 1 1 1
```

MLR Lasso

```
MLRLassoPred = MLRLassoPredFunction(MLRLassoModel,testWHR,testWHRY,parameters = list("lambda"=MLRLassoLa
```

Accuracy.

```
Accuracy(MLRLassoPred,testWHRY)
```

```
## [1] 0.5
```

Confusion Matrix.

```
table(MLRLassoPred,testWHRY)
```

```
##             testWHRY
## MLRLassoPred  3  4  5  6  7  8
##            4  1  2  1  0  0  0
##            5  0  4  4  0  0  0
##            6  1  0  4 10  4  0
##            7  0  0  1  0  1  1
```

MLR Ridge

```
MLRRidgePred = MLRRidgePredFunction(MLRRidgeModel,testWHR,testWHRY,parameters = list("lambda"=MLRRidgeLa
```

Accuracy.

```
Accuracy(MLRRidgePred,testWHRY)
```

```
## [1] 0.5294118
```

Confusion Matrix.

```
table(MLRRidgePred,testWHRY)
```

```
##             testWHRY
## MLRRidgePred 3 4 5 6 7 8
##            4 1 2 0 0 0 0
##            5 0 4 6 0 0 0
##            6 1 0 3 9 4 0
##            7 0 0 1 1 1 1
```

LDA

Accuracy.

```
predictions = model_lda %>% predict(testWHR)
t = table(predictions$class, testWHRY )
print(confusionMatrix(t))
```

```
## Confusion Matrix and Statistics
##
##    testWHRY
##     3 4 5 6 7 8
##   3 0 1 0 0 0 0
##   4 1 5 1 0 0 0
##   5 1 0 5 0 0 0
##   6 0 0 3 9 2 0
##   7 0 0 1 1 3 0
##   8 0 0 0 0 0 1
##
## Overall Statistics
##
##                Accuracy : 0.6765
##                  95% CI : (0.4947, 0.8261)
##     No Information Rate : 0.2941
##     P-Value [Acc > NIR] : 4.547e-06
##
##                   Kappa : 0.5779
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000   0.8333   0.5000   0.9000  0.60000  1.00000
## Specificity           0.96875   0.9286   0.9583   0.7917  0.93103  1.00000
```

```
## Pos Pred Value         0.00000   0.7143   0.8333   0.6429  0.60000  1.00000
## Neg Pred Value         0.93939   0.9630   0.8214   0.9500  0.93103  1.00000
## Prevalence             0.05882   0.1765   0.2941   0.2941  0.14706  0.02941
## Detection Rate         0.00000   0.1471   0.1471   0.2647  0.08824  0.02941
## Detection Prevalence   0.02941   0.2059   0.1765   0.4118  0.14706  0.02941
## Balanced Accuracy      0.48438   0.8810   0.7292   0.8458  0.76552  1.00000
```

As we can see, LDA reached around 70% of accuracy as a classifier. LDA basically projects the data in a new linear feature space, obviously the classifier will reach high accuracy if the data are completly linear separable.

Since LDA assumes normal distributed data, we tried modeling with normalized predictors but the accuracy fell down to 0.2352941 from 0.6764706.

QDA

Accuracy.

```
qda_testWHRY = testWHRY
qda_testWHRY[qda_testWHRY==8] = 7
qda_testWHRY = droplevels(qda_testWHRY)

predictions = model_qda %>% predict(testWHR)
t = table(predictions$class, qda_testWHRY )
print(confusionMatrix(t))
```

```
## Confusion Matrix and Statistics
##
##    qda_testWHRY
##      3  4  5  6  7
##   3  0  1  0  0  0
##   4  2  2  3  0  0
##   5  0  3  5  0  0
##   6  0  0  2 10  4
##   7  0  0  0  0  2
##
## Overall Statistics
##
##                Accuracy : 0.5588
##                  95% CI : (0.3789, 0.7281)
##     No Information Rate : 0.2941
##     P-Value [Acc > NIR] : 0.001137
##
##                   Kappa : 0.407
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7
## Sensitivity           0.00000  0.33333   0.5000   1.0000  0.33333
## Specificity           0.96875  0.82143   0.8750   0.7500  1.00000
## Pos Pred Value        0.00000  0.28571   0.6250   0.6250  1.00000
## Neg Pred Value        0.93939  0.85185   0.8077   1.0000  0.87500
## Prevalence            0.05882  0.17647   0.2941   0.2941  0.17647
## Detection Rate        0.00000  0.05882   0.1471   0.2941  0.05882
```

```
## Detection Prevalence  0.02941  0.20588   0.2353   0.4706  0.05882
## Balanced Accuracy     0.48438  0.57738   0.6875   0.8750  0.66667
```

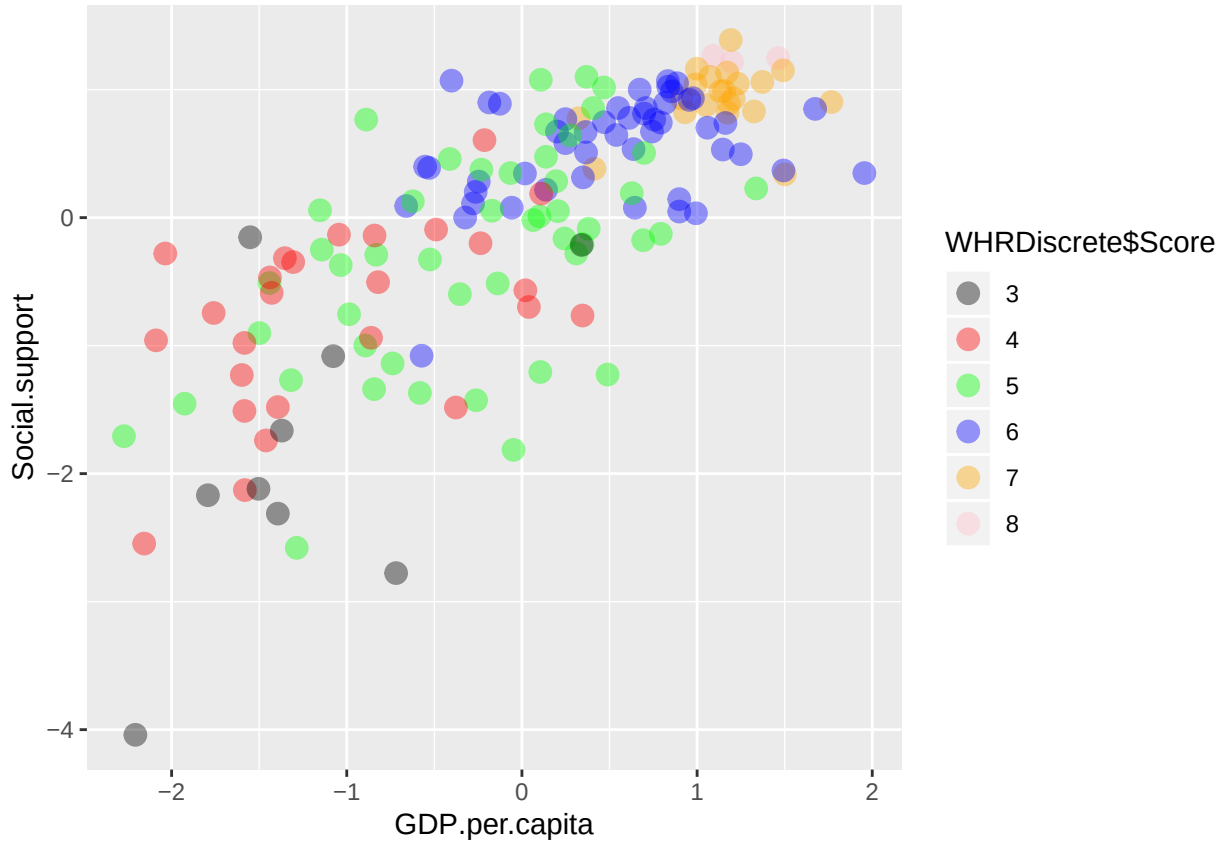
As expected due to the class issue we had to resolve, QDA scored lower in accuracy as a classifier.

Hierarchical clustering

Cluster Plot

All the points where the inner color doesn't match the outer color are the ones which were clustered incorrectly.

```
ggplot(WHRStd, aes(GDP.per.capita, Social.support, color = WHRDiscrete$Score)) +
  geom_point(alpha = 0.4, size = 3.5) + geom_point(col = clusterCut) +
  scale_color_manual(values = c('black', 'red', 'green','blue','orange','pink'))
```



All the points where the inner color doesn't match the outer color are the ones which were clustered incorrectly.

Accuracy is not the most accurate term when it comes to clustering, but so as to see whether the hierarchical clustering gave clusters or groups that coincide with our labels we tried to roughly calculate the coincidence, which tells for each cluster, which is the majority class.

```
Majority_class = tapply(factor(WHRDiscrete$Score),clusterCut,function(i)names(sort(table(i)))[2])
Majority_class
```

```
##   1   2   3
## "3" "6" "6"
```

And from there we tried to see how much this agrees with the actual labels.

```r
mean(Majority_class[clusterCut] == (WHRDiscrete$Score))
```

```
## [1] 0.03846154
```

It was seen that the clustring did not really capture the essence of the data very well.

Results

From all the tested models the linear regression obtained the worst results, of 0.38 accuracy, while the linear discriminant analysis, the best, of 0.68 accuracy. Both LR and LDA placed in their positions by a large margin to the other methods, leaving no doubt, which is the worst and the best method for this data. For both clustering methods we did not obtain predictions and as such accuracy of the methods. Although such would be possible, it would not make sense, given the arbitrary nature of the split of the data into different classes.

Conclusions

The original data, without the division into classes, posed a simple problem, given that there is a known linear deterministic formula to transform the features of the data into the score value. The division of the scores into different classes increases the difficulty of the problem by an extreme amount. Now the linear relations present in the data are much harder to obtain, and as such the performance is substantially lower than what we would previously be able to obtain. To add the problem, the dataset is quite small, not allowing for more moderns methods that rely on large volumes of data (such as neural networks).

Linear regression showed it inadequacy to deal with multiclass classification problems, with all versions of it having much worse results than other methods. Logistic regression performed worse than expected, with all variants achieving only about 0.5 accuracy. While on the other hand LDA performed surprisingly well, showing the strength of this method for classification problems with low amounts of data. There were some problems with QDA, namely the fact that it could not handle the original data, due to the low amount of data for certain classes. This definitely reduced the performance of the model.