covid19\_analysis\_dp.R

danielpearl

2020-06-08

# Load packages  
library(arules)

## Warning: package 'arules' was built under R version 3.6.2

## Loading required package: Matrix

##   
## Attaching package: 'arules'

## The following objects are masked from 'package:base':  
##   
## abbreviate, write

library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(e1071)  
library(FSelector)  
library(klaR)

## Loading required package: MASS

library(rattle)

## Rattle: A free graphical interface for data science with R.  
## Version 5.3.0 Copyright (c) 2006-2018 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.

library(rpart)  
library(rpart.plot)  
library(stringr)  
library(xlsx)  
  
# Read file  
file\_path = "~/Syracuse/03\_Sprint2020/IST707\_Data\_Analysis/project/covid19\_indicators.xlsx"  
indicators = read.xlsx(file\_path, 1)

## Data Pre-processing

# The data has been prepared for two different types of models. The first type contains dependent variables that are categorical with independent variables that are numeric. The training and testing data labeled as c\_indicators and c\_indicators is constructed for the models that require numerical data-types for precision. These models are SVM, KNN, Random Forest, and clustering. For the remaining model, the training and testing data labeled as d\_indicators and d\_test requires all columns to be categorical. This is used to construct a decision tree.  
  
# The first step in data pre-processing involved removing all the columns that were not being used. The following columns were removed from the dataframe: date, new\_cases, new\_deaths, new\_cases\_per\_million, new\_deaths\_per\_million, new\_deaths\_per\_million, new\_tests, new\_tests\_smoothed, new\_tests\_per\_thousand, new\_tests\_smoothed\_per\_thousand, tests\_units, stringency\_index. The next step involved removing the row corresponding to the entire world. Leaving in a column that contained sum data would distort any calculated results from the numeric columns.   
  
# The next step involved discretizing just the dependent variable columns to more easily interpret the results. These columns are: total\_cases\_per\_million and total\_deaths\_per\_million. For most of the discretization, the discretize function from the arules package was used. This function takes in a method, the number of breaks/bins, and the labels. For the dependent variables, 5 bins were created to represent the severity of both the infection rate and the death rate. The method used to divide data into these bins is frequency. By setting the method to frequency, we are ensuring that the values from the columns will be divided up evenly between the bins. This ensures we can maintain a substantial representation within all the bins. The labels that we ended up using were: very low, low, moderate, high, very high.  
  
# For the d\_indicators dataframe, most of the columns are discretized using the same discretize function. Those columns with fewer pieces of data will contain 3 bins while those with more will contain 5. The age column is discretized using the cut function to divide the data into 4 different bins: 10-19, 20-29, 30-39, 40-49. The cut function contains the break parameter which is given a sequence to divide the ages up evenly.  
  
# The last part of preprocessing the data involves replacing the null values. For the c\_indicators dataframe, all the null values within the numeric columns are replaced by the average mean of those columns. For the d\_indicators dataframe, a new bin labeled as ‘unknown’ is created to handle all the null values. In order to do this, the columns are converted to strings where all the null values are filled in with the string ‘unknown’ before they are converted back to factors. The following code is used to do this:  
   
# Remove unnecessary columns  
indicators$date = NULL  
indicators$new\_cases = NULL  
indicators$new\_deaths = NULL  
indicators$new\_cases\_per\_million = NULL  
indicators$new\_deaths\_per\_million = NULL  
indicators$new\_tests = NULL  
indicators$new\_tests\_smoothed = NULL  
indicators$new\_tests\_per\_thousand = NULL  
indicators$new\_tests\_smoothed\_per\_thousand = NULL  
indicators$tests\_units = NULL  
indicators$stringency\_index = NULL  
indicators$total\_cases = NULL  
indicators$total\_deaths = NULL  
indicators$total\_tests = NULL  
  
# Remove World from locations  
indicators = indicators[indicators$iso\_code != "OWID\_WRL",]  
  
# Create labels list for discretization  
x\_labels = c("low", "moderate", "high")  
y\_labels = c("very low", "low", "moderate", "high", "very high")  
  
# Discretize total cases per million   
indicators$total\_cases\_per\_million = discretize(indicators$total\_cases\_per\_million, method = "frequency", breaks = 5, dig.lab = 5, labels=y\_labels)  
table(indicators$total\_cases\_per\_million)

##   
## very low low moderate high very high   
## 42 42 41 42 42

# Discretize total deaths per million   
indicators$total\_deaths\_per\_million = discretize(indicators$total\_deaths\_per\_million, method = "frequency", breaks = 5, dig.lab = 5, labels=y\_labels)  
table(indicators$total\_deaths\_per\_million)

##   
## very low low moderate high very high   
## 42 42 41 42 42

# Divide indicators into discrete and continuous  
c\_indicators = indicators  
d\_indicators = indicators  
  
# Fill null values with the column average  
for(i in 3:ncol(c\_indicators)){  
 c\_indicators[is.na(c\_indicators[,i]), i] = mean(c\_indicators[,i], na.rm = TRUE)}

## Warning in mean.default(c\_indicators[, i], na.rm = TRUE): argument is not  
## numeric or logical: returning NA  
  
## Warning in mean.default(c\_indicators[, i], na.rm = TRUE): argument is not  
## numeric or logical: returning NA

# Set seeds for replication  
#set.seed(550)  
  
# Equal frequency binning used for all numeric values   
  
# Discretize total test  
d\_indicators$total\_tests = discretize(d\_indicators$total\_tests, method = "frequency", breaks = 3, dig.lab = 3, labels=x\_labels)  
table(d\_indicators$total\_tests)

##   
## low moderate high   
## 12 11 12

# Discretize total test per thousand  
d\_indicators$total\_tests\_per\_thousand = discretize(d\_indicators$total\_tests\_per\_thousand, method = "frequency", breaks = 3, dig.lab = 3, labels=x\_labels)  
table(d\_indicators$total\_tests\_per\_thousand)

##   
## low moderate high   
## 12 11 12

# Discretize total test per thousand  
d\_indicators$population = discretize(d\_indicators$population, method = "frequency", breaks = 5, dig.lab = 5, labels=y\_labels)  
table(d\_indicators$population)

##   
## very low low moderate high very high   
## 42 42 41 42 42

# Discretize total test per thousand  
d\_indicators$population\_density = discretize(d\_indicators$population\_density, method = "frequency", breaks = 5, dig.lab = 5, labels=y\_labels)  
table(d\_indicators$population\_density)

##   
## very low low moderate high very high   
## 40 39 39 39 40

# Discretize median age  
d\_indicators$median\_age = cut(d\_indicators$median\_age, breaks = seq(10, 50, by = 10), label=c("10-19", "20-29", "30-39", "40-49"))  
table(d\_indicators$median\_age )

##   
## 10-19 20-29 30-39 40-49   
## 37 56 52 40

# Discretize 65+  
d\_indicators$aged\_65\_older = discretize(d\_indicators$aged\_65\_older, method = "frequency", breaks = 3, dig.lab = 3, labels=x\_labels)  
table(d\_indicators$aged\_65\_older)

##   
## low moderate high   
## 61 60 61

# Discretize 65+  
d\_indicators$aged\_70\_older = discretize(d\_indicators$aged\_70\_older, method = "frequency", breaks = 3, dig.lab = 3, labels=x\_labels)  
table(d\_indicators$aged\_70\_older)

##   
## low moderate high   
## 61 61 62

# Discretize GDP per capita  
d\_indicators$gdp\_per\_capita = discretize(d\_indicators$gdp\_per\_capita, method = "frequency", breaks = 5, dig.lab = 5, labels=y\_labels)  
table(d\_indicators$gdp\_per\_capita)

##   
## very low low moderate high very high   
## 36 36 37 35 37

# Discretize extreme poverty  
d\_indicators$extreme\_poverty = discretize(d\_indicators$extreme\_poverty, method = "frequency", breaks = 3, dig.lab = 3, labels=x\_labels)  
table(d\_indicators$extreme\_poverty)

##   
## low moderate high   
## 38 42 40

# Discretize cvd death rate  
d\_indicators$cvd\_death\_rate = discretize(d\_indicators$cvd\_death\_rate, method = "frequency", breaks = 5, dig.lab = 5, labels=y\_labels)  
table(d\_indicators$cvd\_death\_rate)

##   
## very low low moderate high very high   
## 37 37 37 37 37

# Discretize diabetes prevalence  
d\_indicators$diabetes\_prevalence = discretize(d\_indicators$diabetes\_prevalence, method = "frequency", breaks = 5, dig.lab = 5, labels=y\_labels)  
table(d\_indicators$diabetes\_prevalence)

##   
## very low low moderate high very high   
## 39 38 38 38 39

# Discretize female smokers  
d\_indicators$female\_smokers = discretize(d\_indicators$female\_smokers, method = "frequency", breaks = 3, dig.lab = 3, labels=x\_labels)  
table(d\_indicators$female\_smokers)

##   
## low moderate high   
## 47 46 47

# Discretize male smokers  
d\_indicators$male\_smokers = discretize(d\_indicators$male\_smokers, method = "frequency", breaks = 3, dig.lab = 3, labels=x\_labels)  
table(d\_indicators$male\_smokers)

##   
## low moderate high   
## 43 49 46

# Discretize handwashing facilities  
d\_indicators$handwashing\_facilities = discretize(d\_indicators$handwashing\_facilities, method = "frequency", breaks = 3, dig.lab = 3, labels=x\_labels)  
table(d\_indicators$handwashing\_facilities)

##   
## low moderate high   
## 30 30 31

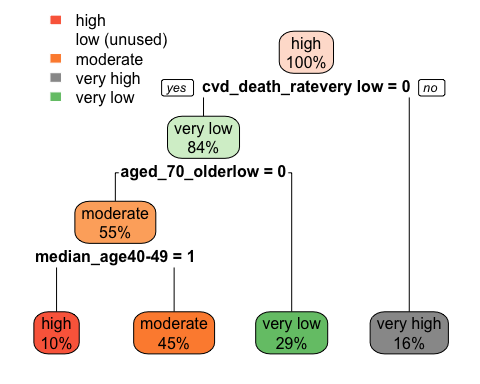
# Discretize hospital beds per 100k  
d\_indicators$hospital\_beds\_per\_100k = discretize(d\_indicators$hospital\_beds\_per\_100k, method = "frequency", breaks = 5, dig.lab = 5, labels=y\_labels)  
table(d\_indicators$hospital\_beds\_per\_100k)

##   
## very low low moderate high very high   
## 33 33 31 34 33

# Fill null values with unknown  
i = sapply(d\_indicators, is.factor)   
d\_indicators[i] = lapply(d\_indicators[i], as.character)   
d\_indicators[is.na(d\_indicators)] = "unknown"  
d\_indicators[i] = lapply(d\_indicators[i], as.factor)   
  
# Partition into training and testing dataframes  
n = nrow(d\_indicators)  
index = sample(n, n\*.7, replace=FALSE)  
# Create training and testing data for discrete datafram  
d\_train = d\_indicators[index,]  
d\_test = d\_indicators[-index,]  
# Create training and testing data for continuous datafram  
c\_train = c\_indicators[index,]  
c\_test = c\_indicators[-index,]  
  
# Define the independent variables  
x\_vars = c("total\_tests\_per\_thousand", "population", "population\_density", "median\_age", "aged\_65\_older", "aged\_70\_older", "gdp\_per\_capita", "extreme\_poverty", "cvd\_death\_rate", "diabetes\_prevalence", "female\_smokers", "male\_smokers", "handwashing\_facilities", "hospital\_beds\_per\_100k")  
  
# This function takes in x and y variables and returns a formula  
create\_formula = function(x,y){  
 f <- as.formula(  
 paste(y,   
 paste(x, collapse = " + "),   
 sep = " ~ "))}

# Decision Trees (DT)

## Decision Tree parameters  
  
# The rpart library is used to create a decision tree model from the R caret package. The model is tuned using the repeated cross validation method. This method is used to tune the model by dividing the data into subsets and repeating a number of times. For this parameter, the data is divided into 10 subsets and repeated 3 times. This is used to prevent the model from overfitting. The metric for optimization is set to accuracy which means that the model will tune iteself until it reaches the most accurate results. The tune length is set to 20 which is the number of tries to reach the optimal value.  
  
# Create formula from the create\_formula function  
formula\_cases = create\_formula(x\_vars, "total\_cases\_per\_million")  
formula\_deaths = create\_formula(x\_vars, "total\_deaths\_per\_million")  
  
# Set cross validation hyper-parameter for models  
tr\_control = trainControl(method = 'repeatedcv', number=10, repeats = 3)  
  
# Build decision tree model for cases per million  
dt\_model\_cases = train(formula\_cases,  
 data=d\_train,  
 method="rpart",  
 metric="Accuracy",  
 trControl=tr\_control,  
 tuneLength = 20  
)  
  
# Visualize Decision Tree  
rpart.plot(dt\_model\_cases$finalModel, extra = 100)



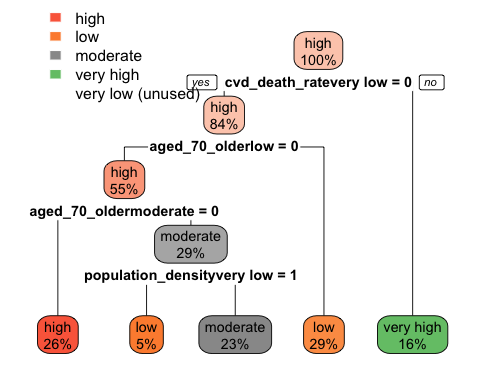
# Prediction for total cases per million  
dt\_pred\_cases = predict(dt\_model\_cases, newdata=d\_test)  
  
# Create confusion matrix for total cases per million  
t <- table(d\_test$total\_cases\_per\_million, dt\_pred\_cases)  
result = confusionMatrix(t)  
result # Show confusion matrix results

## Confusion Matrix and Statistics  
##   
## dt\_pred\_cases  
## high low moderate very high very low  
## high 5 0 4 1 1  
## low 1 0 7 1 5  
## moderate 2 0 4 3 4  
## very high 0 0 5 7 1  
## very low 0 0 4 1 7  
##   
## Overall Statistics  
##   
## Accuracy : 0.3651   
## 95% CI : (0.2473, 0.496)  
## No Information Rate : 0.381   
## P-Value [Acc > NIR] : 0.64798   
##   
## Kappa : 0.2085   
##   
## Mcnemar's Test P-Value : 0.07076   
##   
## Statistics by Class:  
##   
## Class: high Class: low Class: moderate Class: very high  
## Sensitivity 0.62500 NA 0.16667 0.5385  
## Specificity 0.89091 0.7778 0.76923 0.8800  
## Pos Pred Value 0.45455 NA 0.30769 0.5385  
## Neg Pred Value 0.94231 NA 0.60000 0.8800  
## Prevalence 0.12698 0.0000 0.38095 0.2063  
## Detection Rate 0.07937 0.0000 0.06349 0.1111  
## Detection Prevalence 0.17460 0.2222 0.20635 0.2063  
## Balanced Accuracy 0.75795 NA 0.46795 0.7092  
## Class: very low  
## Sensitivity 0.3889  
## Specificity 0.8889  
## Pos Pred Value 0.5833  
## Neg Pred Value 0.7843  
## Prevalence 0.2857  
## Detection Rate 0.1111  
## Detection Prevalence 0.1905  
## Balanced Accuracy 0.6389

result[-1] # Show results divided into classes

## $table  
## dt\_pred\_cases  
## high low moderate very high very low  
## high 5 0 4 1 1  
## low 1 0 7 1 5  
## moderate 2 0 4 3 4  
## very high 0 0 5 7 1  
## very low 0 0 4 1 7  
##   
## $overall  
## Accuracy Kappa AccuracyLower AccuracyUpper AccuracyNull   
## 0.36507937 0.20854271 0.24734326 0.49595122 0.38095238   
## AccuracyPValue McnemarPValue   
## 0.64797515 0.07075647   
##   
## $byClass  
## Sensitivity Specificity Pos Pred Value Neg Pred Value  
## Class: high 0.6250000 0.8909091 0.4545455 0.9423077  
## Class: low NA 0.7777778 NA NA  
## Class: moderate 0.1666667 0.7692308 0.3076923 0.6000000  
## Class: very high 0.5384615 0.8800000 0.5384615 0.8800000  
## Class: very low 0.3888889 0.8888889 0.5833333 0.7843137  
## Precision Recall F1 Prevalence Detection Rate  
## Class: high 0.4545455 0.6250000 0.5263158 0.1269841 0.07936508  
## Class: low 0.0000000 NA NA 0.0000000 0.00000000  
## Class: moderate 0.3076923 0.1666667 0.2162162 0.3809524 0.06349206  
## Class: very high 0.5384615 0.5384615 0.5384615 0.2063492 0.11111111  
## Class: very low 0.5833333 0.3888889 0.4666667 0.2857143 0.11111111  
## Detection Prevalence Balanced Accuracy  
## Class: high 0.1746032 0.7579545  
## Class: low 0.2222222 NA  
## Class: moderate 0.2063492 0.4679487  
## Class: very high 0.2063492 0.7092308  
## Class: very low 0.1904762 0.6388889  
##   
## $mode  
## [1] "sens\_spec"  
##   
## $dots  
## list()

# Build decision tree model for deaths per million  
dt\_model\_deaths = train(formula\_deaths,  
 data=d\_train,  
 method="rpart",  
 metric="Accuracy",  
 trControl=tr\_control,  
 tuneLength = 20  
 )  
  
# Visualize Decision Tree  
rpart.plot(dt\_model\_deaths$finalModel, extra = 100)



# Prediction for total cases per million  
dt\_pred\_deaths = predict(dt\_model\_deaths, newdata=d\_test)  
  
# Create confusion matrix for total cases per million  
t <- table(d\_test$total\_cases\_per\_million, dt\_pred\_deaths)  
result = confusionMatrix(t)  
result # Show confusion matrix results

## Confusion Matrix and Statistics  
##   
## dt\_pred\_deaths  
## high low moderate very high very low  
## high 7 2 1 1 0  
## low 2 6 5 1 0  
## moderate 5 4 1 3 0  
## very high 5 1 0 7 0  
## very low 0 8 3 1 0  
##   
## Overall Statistics  
##   
## Accuracy : 0.3333   
## 95% CI : (0.2195, 0.4634)  
## No Information Rate : 0.3333   
## P-Value [Acc > NIR] : 0.5473   
##   
## Kappa : 0.1645   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: high Class: low Class: moderate Class: very high  
## Sensitivity 0.3684 0.28571 0.10000 0.5385  
## Specificity 0.9091 0.80952 0.77358 0.8800  
## Pos Pred Value 0.6364 0.42857 0.07692 0.5385  
## Neg Pred Value 0.7692 0.69388 0.82000 0.8800  
## Prevalence 0.3016 0.33333 0.15873 0.2063  
## Detection Rate 0.1111 0.09524 0.01587 0.1111  
## Detection Prevalence 0.1746 0.22222 0.20635 0.2063  
## Balanced Accuracy 0.6388 0.54762 0.43679 0.7092  
## Class: very low  
## Sensitivity NA  
## Specificity 0.8095  
## Pos Pred Value NA  
## Neg Pred Value NA  
## Prevalence 0.0000  
## Detection Rate 0.0000  
## Detection Prevalence 0.1905  
## Balanced Accuracy NA

result[-1] # Show results divided into classes

## $table  
## dt\_pred\_deaths  
## high low moderate very high very low  
## high 7 2 1 1 0  
## low 2 6 5 1 0  
## moderate 5 4 1 3 0  
## very high 5 1 0 7 0  
## very low 0 8 3 1 0  
##   
## $overall  
## Accuracy Kappa AccuracyLower AccuracyUpper AccuracyNull   
## 0.3333333 0.1645090 0.2195375 0.4633808 0.3333333   
## AccuracyPValue McnemarPValue   
## 0.5472513 NaN   
##   
## $byClass  
## Sensitivity Specificity Pos Pred Value Neg Pred Value  
## Class: high 0.3684211 0.9090909 0.63636364 0.7692308  
## Class: low 0.2857143 0.8095238 0.42857143 0.6938776  
## Class: moderate 0.1000000 0.7735849 0.07692308 0.8200000  
## Class: very high 0.5384615 0.8800000 0.53846154 0.8800000  
## Class: very low NA 0.8095238 NA NA  
## Precision Recall F1 Prevalence Detection Rate  
## Class: high 0.63636364 0.3684211 0.46666667 0.3015873 0.11111111  
## Class: low 0.42857143 0.2857143 0.34285714 0.3333333 0.09523810  
## Class: moderate 0.07692308 0.1000000 0.08695652 0.1587302 0.01587302  
## Class: very high 0.53846154 0.5384615 0.53846154 0.2063492 0.11111111  
## Class: very low 0.00000000 NA NA 0.0000000 0.00000000  
## Detection Prevalence Balanced Accuracy  
## Class: high 0.1746032 0.6387560  
## Class: low 0.2222222 0.5476190  
## Class: moderate 0.2063492 0.4367925  
## Class: very high 0.2063492 0.7092308  
## Class: very low 0.1904762 NA  
##   
## $mode  
## [1] "sens\_spec"  
##   
## $dots  
## list()

# Combine the predictions with countries.  
country = data.frame(d\_test$location)  
cases = data.frame(dt\_pred\_cases)  
deaths = data.frame(dt\_pred\_deaths)  
covid19\_dt\_pred = cbind(country, cases, deaths)  
  
# Change column names  
labels = c("country", "cases", "deaths")  
colnames(covid19\_dt\_pred) = labels  
  
# Write prediction dataframe to csv file  
write.csv(covid19\_dt\_pred, 'decision\_tree\_pred.csv')  
  
## Decision Tree Analysis  
  
# The top indicators for the infection rate are handwashing facilities, gdp\_per\_capita, cvd death rate, median age between 30-49, age percentage over 65, and hospital beds. Some of the node connections seem intuitive. For example countries with a lower gdp per capita and a higher cardio vascular death rate have a high infection rate. Though we're assuming most of these indicators lead to a change in infection rate, it's possible that the infection rate might lead to a number of changes in indicators. One example of this is the number of hospital beds. Rather than assume that the number of hospital beds are affecting the infection rate, it might also be reasonable to suggest that a higher infection rate might lead to a higher number of hospital beds.   
  
# The top indicators for the death rate are percentage of population older than 70, population density, hospital beds, and gdp per capita. Age seems to be correlated with the number of deaths. Countries with a low percentage of elderly people over the age of 70 have a lower death rate. Countries with a higher population density also have a higher death rate. Fewer hospital beds seem to indicate a higher death rate as well. It's possible that the relationship between the y-variable rates and the x-variable (hospital beds) are inverted between the two models.   
  
# Both the infection and death rate decision tree models performed poorly with an accuracy of around 30-45%. In addition to this, both the precision and recall of these models performed far under 50%. One explanation for this might be the small dataset size. A dataset that showed the demographic details for every individual infected would be far more valuable and accurate. Since this data only shows the summary in each country, various unknown cultural phenomenon might also be influencing both dependent variables.

# Support Vector Machines (SVM)

# The SVM model uses an algorithm that can solve both linearly separable and inseparable problems. The goal is to find a linear hyperplane that can separate the data and maximize the margin between the data points.  
  
# The method used to build this model is svmRadial. This is the shape for the kernel. The SVM model will use the trControl parameter to tune itself using the same repeated cross validation defined when creating the decision tree. The cost hyperparameter represented by C in tune\_grid corresponds to the number of training errors allowed. Sigma is the width of the Gaussian distribution. The model will iterate through values of C and sigma between 0 and 1. The hyper-parameter preProcess is set to center and scale. This is used to estimate the location and scale of the predictors.   
  
# Set hyper-parameters for svm model  
tune\_grid = expand.grid(sigma = seq(0, 1, 0.1), C = c(0.1, 0.2, 0.3, 0.4, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1))  
  
# Build svm model for cases per million  
svm\_model\_cases = train(formula\_cases,   
 data=c\_train,   
 method = 'svmRadial',   
 metric="Accuracy",  
 preProcess = c('center', 'scale'),   
 trControl = tr\_control,   
 tuneLength = 10,  
 tuneGrid=tune\_grid)  
   
# Prediction for total cases per million  
svm\_pred\_cases = predict(svm\_model\_cases, newdata=c\_test)  
  
# Create confusion matrix for total cases per million  
t <- table(c\_test$total\_cases\_per\_million, svm\_pred\_cases)  
result = confusionMatrix(t)  
result # Show confusion matrix results

## Confusion Matrix and Statistics  
##   
## svm\_pred\_cases  
## very low low moderate high very high  
## very low 8 2 1 0 1  
## low 6 3 3 1 1  
## moderate 3 0 4 3 3  
## high 2 0 3 5 1  
## very high 0 1 4 0 8  
##   
## Overall Statistics  
##   
## Accuracy : 0.4444   
## 95% CI : (0.3192, 0.5751)  
## No Information Rate : 0.3016   
## P-Value [Acc > NIR] : 0.01154   
##   
## Kappa : 0.3068   
##   
## Mcnemar's Test P-Value : 0.34650   
##   
## Statistics by Class:  
##   
## Class: very low Class: low Class: moderate Class: high  
## Sensitivity 0.4211 0.50000 0.26667 0.55556  
## Specificity 0.9091 0.80702 0.81250 0.88889  
## Pos Pred Value 0.6667 0.21429 0.30769 0.45455  
## Neg Pred Value 0.7843 0.93878 0.78000 0.92308  
## Prevalence 0.3016 0.09524 0.23810 0.14286  
## Detection Rate 0.1270 0.04762 0.06349 0.07937  
## Detection Prevalence 0.1905 0.22222 0.20635 0.17460  
## Balanced Accuracy 0.6651 0.65351 0.53958 0.72222  
## Class: very high  
## Sensitivity 0.5714  
## Specificity 0.8980  
## Pos Pred Value 0.6154  
## Neg Pred Value 0.8800  
## Prevalence 0.2222  
## Detection Rate 0.1270  
## Detection Prevalence 0.2063  
## Balanced Accuracy 0.7347

result[-1] # Show results divided into classes

## $table  
## svm\_pred\_cases  
## very low low moderate high very high  
## very low 8 2 1 0 1  
## low 6 3 3 1 1  
## moderate 3 0 4 3 3  
## high 2 0 3 5 1  
## very high 0 1 4 0 8  
##   
## $overall  
## Accuracy Kappa AccuracyLower AccuracyUpper AccuracyNull   
## 0.44444444 0.30682175 0.31917313 0.57511237 0.30158730   
## AccuracyPValue McnemarPValue   
## 0.01154176 0.34649702   
##   
## $byClass  
## Sensitivity Specificity Pos Pred Value Neg Pred Value  
## Class: very low 0.4210526 0.9090909 0.6666667 0.7843137  
## Class: low 0.5000000 0.8070175 0.2142857 0.9387755  
## Class: moderate 0.2666667 0.8125000 0.3076923 0.7800000  
## Class: high 0.5555556 0.8888889 0.4545455 0.9230769  
## Class: very high 0.5714286 0.8979592 0.6153846 0.8800000  
## Precision Recall F1 Prevalence Detection Rate  
## Class: very low 0.6666667 0.4210526 0.5161290 0.3015873 0.12698413  
## Class: low 0.2142857 0.5000000 0.3000000 0.0952381 0.04761905  
## Class: moderate 0.3076923 0.2666667 0.2857143 0.2380952 0.06349206  
## Class: high 0.4545455 0.5555556 0.5000000 0.1428571 0.07936508  
## Class: very high 0.6153846 0.5714286 0.5925926 0.2222222 0.12698413  
## Detection Prevalence Balanced Accuracy  
## Class: very low 0.1904762 0.6650718  
## Class: low 0.2222222 0.6535088  
## Class: moderate 0.2063492 0.5395833  
## Class: high 0.1746032 0.7222222  
## Class: very high 0.2063492 0.7346939  
##   
## $mode  
## [1] "sens\_spec"  
##   
## $dots  
## list()

# Build svm model for deaths per million  
svm\_model\_deaths = train(formula\_deaths,   
 data=c\_train,   
 method = 'svmRadial',   
 metric="Accuracy",  
 preProcess = c('center', 'scale'),   
 trControl = tr\_control,   
 tuneLength = 10,  
 tuneGrid=tune\_grid)  
  
# Prediction for total cases per million  
svm\_pred\_deaths = predict(svm\_model\_deaths, newdata=c\_test)  
  
# Create confusion matrix for total deaths per million  
t <- table(c\_test$total\_deaths\_per\_million, svm\_pred\_deaths)  
result = confusionMatrix(t)  
result # Show confusion matrix results

## Confusion Matrix and Statistics  
##   
## svm\_pred\_deaths  
## very low low moderate high very high  
## very low 2 7 2 5 0  
## low 2 5 3 1 0  
## moderate 0 6 4 4 1  
## high 0 0 0 5 3  
## very high 0 1 0 9 3  
##   
## Overall Statistics  
##   
## Accuracy : 0.3016   
## 95% CI : (0.1923, 0.4302)  
## No Information Rate : 0.381   
## P-Value [Acc > NIR] : 0.9251   
##   
## Kappa : 0.1544   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: very low Class: low Class: moderate Class: high  
## Sensitivity 0.50000 0.26316 0.44444 0.20833  
## Specificity 0.76271 0.86364 0.79630 0.92308  
## Pos Pred Value 0.12500 0.45455 0.26667 0.62500  
## Neg Pred Value 0.95745 0.73077 0.89583 0.65455  
## Prevalence 0.06349 0.30159 0.14286 0.38095  
## Detection Rate 0.03175 0.07937 0.06349 0.07937  
## Detection Prevalence 0.25397 0.17460 0.23810 0.12698  
## Balanced Accuracy 0.63136 0.56340 0.62037 0.56571  
## Class: very high  
## Sensitivity 0.42857  
## Specificity 0.82143  
## Pos Pred Value 0.23077  
## Neg Pred Value 0.92000  
## Prevalence 0.11111  
## Detection Rate 0.04762  
## Detection Prevalence 0.20635  
## Balanced Accuracy 0.62500

result[-1] # Show results divided into classes

## $table  
## svm\_pred\_deaths  
## very low low moderate high very high  
## very low 2 7 2 5 0  
## low 2 5 3 1 0  
## moderate 0 6 4 4 1  
## high 0 0 0 5 3  
## very high 0 1 0 9 3  
##   
## $overall  
## Accuracy Kappa AccuracyLower AccuracyUpper AccuracyNull   
## 0.3015873 0.1543624 0.1923106 0.4302498 0.3809524   
## AccuracyPValue McnemarPValue   
## 0.9251171 NaN   
##   
## $byClass  
## Sensitivity Specificity Pos Pred Value Neg Pred Value  
## Class: very low 0.5000000 0.7627119 0.1250000 0.9574468  
## Class: low 0.2631579 0.8636364 0.4545455 0.7307692  
## Class: moderate 0.4444444 0.7962963 0.2666667 0.8958333  
## Class: high 0.2083333 0.9230769 0.6250000 0.6545455  
## Class: very high 0.4285714 0.8214286 0.2307692 0.9200000  
## Precision Recall F1 Prevalence Detection Rate  
## Class: very low 0.1250000 0.5000000 0.2000000 0.06349206 0.03174603  
## Class: low 0.4545455 0.2631579 0.3333333 0.30158730 0.07936508  
## Class: moderate 0.2666667 0.4444444 0.3333333 0.14285714 0.06349206  
## Class: high 0.6250000 0.2083333 0.3125000 0.38095238 0.07936508  
## Class: very high 0.2307692 0.4285714 0.3000000 0.11111111 0.04761905  
## Detection Prevalence Balanced Accuracy  
## Class: very low 0.2539683 0.6313559  
## Class: low 0.1746032 0.5633971  
## Class: moderate 0.2380952 0.6203704  
## Class: high 0.1269841 0.5657051  
## Class: very high 0.2063492 0.6250000  
##   
## $mode  
## [1] "sens\_spec"  
##   
## $dots  
## list()

# Combine the predictions with countries.  
country = data.frame(c\_test$location)  
cases = data.frame(svm\_pred\_cases)  
deaths = data.frame(svm\_pred\_deaths)  
covid19\_svm\_pred = cbind(country, cases, deaths)  
  
# Change column names  
labels = c("country", "cases", "deaths")  
colnames(covid19\_svm\_pred) = labels  
  
# Write prediction dataframe to csv file  
write.csv(covid19\_svm\_pred, 'svm\_pred.csv')  
  
## SVM Analysis  
  
# The confusion matrix shows an Accuracy of 40-50% for both the infection and death rate which indicates that this model is better at predicting than the decision tree. The precision, recall, and F-measure are fluctuate quite a bit. From looking at the separate classes, it looks like the infection rate extremities are the easiest to predict with a balanced accuracy of 74% for very low rates and 66.6% for very high rates. For the death rate, the two highest measures are the easiest to predict with a balanced accuracy of 69.6% for high rates and 82.6% for very high rates.

# Random Forest (RF)

# The random forest algorithm creates random vectors to build multiple decision trees which are ultimately combined together. It is a type of ensemble learning algorithm.  
  
# The tune grid hyper-parameter uses bagging to create a training model that replaces all the values in the full training sample set. It will randomly pick a small set of attributes to build each base model. It will then choose the square root of the original feature size and assign random values from the original dataset. Again the center and scale parameter is set to center and scale and the repeated cross validation control is used to tune the model. The rf method is used with the caret package.  
  
# Set hyper-parameters for rf models  
tune\_grid = expand.grid(.mtry = c(sqrt(ncol(d\_train))))  
  
# Build rf model for cases per million  
rf\_model\_cases = train(formula\_deaths,   
 data=c\_train,   
 method = 'rf',   
 metric="Accuracy",  
 preProcess = c('center', 'scale'),   
 trControl = tr\_control,   
 tuneLength = 10,   
 tuneGrid = tune\_grid)  
  
# Prediction for for cases per million  
rf\_pred\_cases = predict(rf\_model\_cases, c\_test)  
  
# Create confusion matrix for total cases  
t = table(c\_test$total\_cases\_per\_million, rf\_pred\_cases)  
rf\_result = confusionMatrix(t)  
rf\_result

## Confusion Matrix and Statistics  
##   
## rf\_pred\_cases  
## very low low moderate high very high  
## very low 4 6 0 1 1  
## low 0 7 3 2 2  
## moderate 3 3 0 4 3  
## high 1 1 2 2 5  
## very high 5 0 0 2 6  
##   
## Overall Statistics  
##   
## Accuracy : 0.3016   
## 95% CI : (0.1923, 0.4302)  
## No Information Rate : 0.2698   
## P-Value [Acc > NIR] : 0.32881   
##   
## Kappa : 0.125   
##   
## Mcnemar's Test P-Value : 0.04087   
##   
## Statistics by Class:  
##   
## Class: very low Class: low Class: moderate Class: high  
## Sensitivity 0.30769 0.4118 0.00000 0.18182  
## Specificity 0.84000 0.8478 0.77586 0.82692  
## Pos Pred Value 0.33333 0.5000 0.00000 0.18182  
## Neg Pred Value 0.82353 0.7959 0.90000 0.82692  
## Prevalence 0.20635 0.2698 0.07937 0.17460  
## Detection Rate 0.06349 0.1111 0.00000 0.03175  
## Detection Prevalence 0.19048 0.2222 0.20635 0.17460  
## Balanced Accuracy 0.57385 0.6298 0.38793 0.50437  
## Class: very high  
## Sensitivity 0.35294  
## Specificity 0.84783  
## Pos Pred Value 0.46154  
## Neg Pred Value 0.78000  
## Prevalence 0.26984  
## Detection Rate 0.09524  
## Detection Prevalence 0.20635  
## Balanced Accuracy 0.60038

rf\_result[-1] # Show results divided into classes

## $table  
## rf\_pred\_cases  
## very low low moderate high very high  
## very low 4 6 0 1 1  
## low 0 7 3 2 2  
## moderate 3 3 0 4 3  
## high 1 1 2 2 5  
## very high 5 0 0 2 6  
##   
## $overall  
## Accuracy Kappa AccuracyLower AccuracyUpper AccuracyNull   
## 0.3015873 0.1250000 0.1923106 0.4302498 0.2698413   
## AccuracyPValue McnemarPValue   
## 0.3288135 0.0408717   
##   
## $byClass  
## Sensitivity Specificity Pos Pred Value Neg Pred Value  
## Class: very low 0.3076923 0.8400000 0.3333333 0.8235294  
## Class: low 0.4117647 0.8478261 0.5000000 0.7959184  
## Class: moderate 0.0000000 0.7758621 0.0000000 0.9000000  
## Class: high 0.1818182 0.8269231 0.1818182 0.8269231  
## Class: very high 0.3529412 0.8478261 0.4615385 0.7800000  
## Precision Recall F1 Prevalence Detection Rate  
## Class: very low 0.3333333 0.3076923 0.3200000 0.20634921 0.06349206  
## Class: low 0.5000000 0.4117647 0.4516129 0.26984127 0.11111111  
## Class: moderate 0.0000000 0.0000000 NaN 0.07936508 0.00000000  
## Class: high 0.1818182 0.1818182 0.1818182 0.17460317 0.03174603  
## Class: very high 0.4615385 0.3529412 0.4000000 0.26984127 0.09523810  
## Detection Prevalence Balanced Accuracy  
## Class: very low 0.1904762 0.5738462  
## Class: low 0.2222222 0.6297954  
## Class: moderate 0.2063492 0.3879310  
## Class: high 0.1746032 0.5043706  
## Class: very high 0.2063492 0.6003836  
##   
## $mode  
## [1] "sens\_spec"  
##   
## $dots  
## list()

# Build rf model for deaths per million  
rf\_model\_deaths = train(formula\_deaths,   
 data=c\_train,   
 method = 'rf',   
 metric="Accuracy",  
 preProcess = c('center', 'scale'),   
 trControl = tr\_control,   
 tuneLength = 10,   
 tuneGrid = tune\_grid)  
  
# Prediction for cases per million  
rf\_pred\_deaths = predict(rf\_model\_deaths, c\_test)  
  
# Create confusion matrix for total cases  
t = table(c\_test$total\_cases\_per\_million, rf\_pred\_deaths)  
rf\_result = confusionMatrix(t)  
rf\_result

## Confusion Matrix and Statistics  
##   
## rf\_pred\_deaths  
## very low low moderate high very high  
## very low 4 6 0 1 1  
## low 0 7 3 2 2  
## moderate 3 3 0 5 2  
## high 2 2 0 2 5  
## very high 5 0 0 3 5  
##   
## Overall Statistics  
##   
## Accuracy : 0.2857   
## 95% CI : (0.1789, 0.4135)  
## No Information Rate : 0.2857   
## P-Value [Acc > NIR] : 0.54756   
##   
## Kappa : 0.1062   
##   
## Mcnemar's Test P-Value : 0.01786   
##   
## Statistics by Class:  
##   
## Class: very low Class: low Class: moderate Class: high  
## Sensitivity 0.28571 0.3889 0.00000 0.15385  
## Specificity 0.83673 0.8444 0.78333 0.82000  
## Pos Pred Value 0.33333 0.5000 0.00000 0.18182  
## Neg Pred Value 0.80392 0.7755 0.94000 0.78846  
## Prevalence 0.22222 0.2857 0.04762 0.20635  
## Detection Rate 0.06349 0.1111 0.00000 0.03175  
## Detection Prevalence 0.19048 0.2222 0.20635 0.17460  
## Balanced Accuracy 0.56122 0.6167 0.39167 0.48692  
## Class: very high  
## Sensitivity 0.33333  
## Specificity 0.83333  
## Pos Pred Value 0.38462  
## Neg Pred Value 0.80000  
## Prevalence 0.23810  
## Detection Rate 0.07937  
## Detection Prevalence 0.20635  
## Balanced Accuracy 0.58333

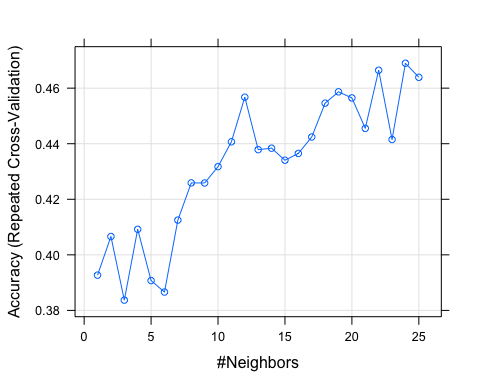
rf\_result[-1] # Show results divided into classes

## $table  
## rf\_pred\_deaths  
## very low low moderate high very high  
## very low 4 6 0 1 1  
## low 0 7 3 2 2  
## moderate 3 3 0 5 2  
## high 2 2 0 2 5  
## very high 5 0 0 3 5  
##   
## $overall  
## Accuracy Kappa AccuracyLower AccuracyUpper AccuracyNull   
## 0.28571429 0.10624212 0.17892921 0.41346103 0.28571429   
## AccuracyPValue McnemarPValue   
## 0.54755713 0.01786459   
##   
## $byClass  
## Sensitivity Specificity Pos Pred Value Neg Pred Value  
## Class: very low 0.2857143 0.8367347 0.3333333 0.8039216  
## Class: low 0.3888889 0.8444444 0.5000000 0.7755102  
## Class: moderate 0.0000000 0.7833333 0.0000000 0.9400000  
## Class: high 0.1538462 0.8200000 0.1818182 0.7884615  
## Class: very high 0.3333333 0.8333333 0.3846154 0.8000000  
## Precision Recall F1 Prevalence Detection Rate  
## Class: very low 0.3333333 0.2857143 0.3076923 0.22222222 0.06349206  
## Class: low 0.5000000 0.3888889 0.4375000 0.28571429 0.11111111  
## Class: moderate 0.0000000 0.0000000 NaN 0.04761905 0.00000000  
## Class: high 0.1818182 0.1538462 0.1666667 0.20634921 0.03174603  
## Class: very high 0.3846154 0.3333333 0.3571429 0.23809524 0.07936508  
## Detection Prevalence Balanced Accuracy  
## Class: very low 0.1904762 0.5612245  
## Class: low 0.2222222 0.6166667  
## Class: moderate 0.2063492 0.3916667  
## Class: high 0.1746032 0.4869231  
## Class: very high 0.2063492 0.5833333  
##   
## $mode  
## [1] "sens\_spec"  
##   
## $dots  
## list()

# Combine the predictions with countries.  
country = data.frame(c\_test$location)  
cases = data.frame(rf\_pred\_cases)  
deaths = data.frame(rf\_pred\_deaths)  
covid19\_rf\_pred = cbind(country, cases, deaths)  
  
# Change column names  
labels = c("country", "cases", "deaths")  
colnames(covid19\_rf\_pred) = labels  
  
# Write prediction dataframe to csv file  
write.csv(covid19\_rf\_pred, 'random\_forest\_pred.csv')  
  
## Random Forest Analysis  
  
# The confusion matrix shows an Accuracy of around 30% which indicates that this is a poor model for predicting the labels. The precision and recall also have a value around 30%. From looking at the separate classes, it also appears that the extremities (very low and very high) are the easiest to predict for the infection rate and the two highest (high and very high) are the easiest to predict for the death rate.

# K-Nearest Neighbors (K-NN)

# K-NN analysis is instance based learning which stores the training examples without calculating anything during the training process. Classification and prediction are delayed until new examples are given.  
  
# The tune grid hyperparameter is set to iterate between 1 and 25. Setting tune grid defines the data points (k). A higher value (more data points) for k is more likely to increase the likelihood for different classes to be included that shouldn't be included. The downside is an increase of bias and a decrease of variance. If the model is underfit the stability will be low and the bias will be high. A K-NN Overfitted model has a low bias and higher variance.   
  
# Set hyper-parameters for knn model  
tune\_grid = expand.grid(k=seq(1, 25, by=1))  
  
# Create knn model for cases per million  
knn\_model\_cases = train(formula\_cases,   
 data = c\_train,   
 method = 'knn',   
 preProcess = c('center', 'scale'),   
 trControl = tr\_control,   
 metric="Accuracy",   
 tuneLength = 10,   
 tuneGrid = tune\_grid)  
plot(knn\_model\_cases)



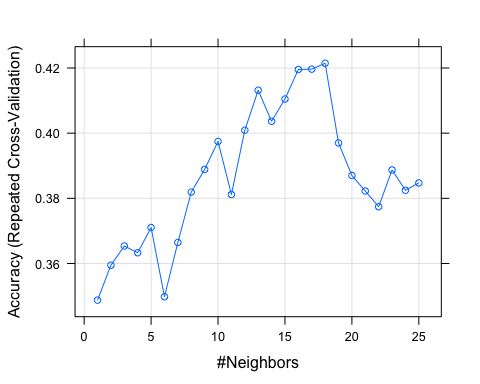
# Prediction for cases per million  
knn\_pred\_cases = predict(knn\_model\_cases, c\_test)  
t <- table(c\_test$total\_cases\_per\_million, knn\_pred\_cases)  
knn\_result = confusionMatrix(t)  
knn\_result

## Confusion Matrix and Statistics  
##   
## knn\_pred\_cases  
## very low low moderate high very high  
## very low 8 1 3 0 0  
## low 7 2 4 1 0  
## moderate 3 0 5 3 2  
## high 1 0 3 6 1  
## very high 0 0 6 0 7  
##   
## Overall Statistics  
##   
## Accuracy : 0.4444   
## 95% CI : (0.3192, 0.5751)  
## No Information Rate : 0.3333   
## P-Value [Acc > NIR] : 0.04331   
##   
## Kappa : 0.3079   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: very low Class: low Class: moderate Class: high  
## Sensitivity 0.4211 0.66667 0.23810 0.60000  
## Specificity 0.9091 0.80000 0.80952 0.90566  
## Pos Pred Value 0.6667 0.14286 0.38462 0.54545  
## Neg Pred Value 0.7843 0.97959 0.68000 0.92308  
## Prevalence 0.3016 0.04762 0.33333 0.15873  
## Detection Rate 0.1270 0.03175 0.07937 0.09524  
## Detection Prevalence 0.1905 0.22222 0.20635 0.17460  
## Balanced Accuracy 0.6651 0.73333 0.52381 0.75283  
## Class: very high  
## Sensitivity 0.7000  
## Specificity 0.8868  
## Pos Pred Value 0.5385  
## Neg Pred Value 0.9400  
## Prevalence 0.1587  
## Detection Rate 0.1111  
## Detection Prevalence 0.2063  
## Balanced Accuracy 0.7934

knn\_result[-1] # Show results divided into classes

## $table  
## knn\_pred\_cases  
## very low low moderate high very high  
## very low 8 1 3 0 0  
## low 7 2 4 1 0  
## moderate 3 0 5 3 2  
## high 1 0 3 6 1  
## very high 0 0 6 0 7  
##   
## $overall  
## Accuracy Kappa AccuracyLower AccuracyUpper AccuracyNull   
## 0.44444444 0.30790960 0.31917313 0.57511237 0.33333333   
## AccuracyPValue McnemarPValue   
## 0.04331004 NaN   
##   
## $byClass  
## Sensitivity Specificity Pos Pred Value Neg Pred Value  
## Class: very low 0.4210526 0.9090909 0.6666667 0.7843137  
## Class: low 0.6666667 0.8000000 0.1428571 0.9795918  
## Class: moderate 0.2380952 0.8095238 0.3846154 0.6800000  
## Class: high 0.6000000 0.9056604 0.5454545 0.9230769  
## Class: very high 0.7000000 0.8867925 0.5384615 0.9400000  
## Precision Recall F1 Prevalence Detection Rate  
## Class: very low 0.6666667 0.4210526 0.5161290 0.30158730 0.12698413  
## Class: low 0.1428571 0.6666667 0.2352941 0.04761905 0.03174603  
## Class: moderate 0.3846154 0.2380952 0.2941176 0.33333333 0.07936508  
## Class: high 0.5454545 0.6000000 0.5714286 0.15873016 0.09523810  
## Class: very high 0.5384615 0.7000000 0.6086957 0.15873016 0.11111111  
## Detection Prevalence Balanced Accuracy  
## Class: very low 0.1904762 0.6650718  
## Class: low 0.2222222 0.7333333  
## Class: moderate 0.2063492 0.5238095  
## Class: high 0.1746032 0.7528302  
## Class: very high 0.2063492 0.7933962  
##   
## $mode  
## [1] "sens\_spec"  
##   
## $dots  
## list()

# Create knn model for deaths per million  
knn\_model\_deaths = train(formula\_deaths,   
 data = c\_train,   
 method = 'knn',   
 preProcess = c('center', 'scale'),   
 trControl = tr\_control,   
 metric="Accuracy",   
 tuneLength = 10,   
 tuneGrid = tune\_grid)  
plot(knn\_model\_deaths)



# Prediction for deaths per million  
knn\_pred\_deaths = predict(knn\_model\_deaths, c\_test)  
t <- table(c\_test$total\_deaths\_per\_million, knn\_pred\_deaths)  
knn\_result = confusionMatrix(t)  
knn\_result

## Confusion Matrix and Statistics  
##   
## knn\_pred\_deaths  
## very low low moderate high very high  
## very low 7 5 1 3 0  
## low 2 6 2 1 0  
## moderate 4 2 1 4 4  
## high 1 0 0 2 5  
## very high 4 0 0 5 4  
##   
## Overall Statistics  
##   
## Accuracy : 0.3175   
## 95% CI : (0.2058, 0.4469)  
## No Information Rate : 0.2857   
## P-Value [Acc > NIR] : 0.332   
##   
## Kappa : 0.1505   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: very low Class: low Class: moderate Class: high  
## Sensitivity 0.3889 0.46154 0.25000 0.13333  
## Specificity 0.8000 0.90000 0.76271 0.87500  
## Pos Pred Value 0.4375 0.54545 0.06667 0.25000  
## Neg Pred Value 0.7660 0.86538 0.93750 0.76364  
## Prevalence 0.2857 0.20635 0.06349 0.23810  
## Detection Rate 0.1111 0.09524 0.01587 0.03175  
## Detection Prevalence 0.2540 0.17460 0.23810 0.12698  
## Balanced Accuracy 0.5944 0.68077 0.50636 0.50417  
## Class: very high  
## Sensitivity 0.30769  
## Specificity 0.82000  
## Pos Pred Value 0.30769  
## Neg Pred Value 0.82000  
## Prevalence 0.20635  
## Detection Rate 0.06349  
## Detection Prevalence 0.20635  
## Balanced Accuracy 0.56385

knn\_result[-1] # Show results divided into classes

## $table  
## knn\_pred\_deaths  
## very low low moderate high very high  
## very low 7 5 1 3 0  
## low 2 6 2 1 0  
## moderate 4 2 1 4 4  
## high 1 0 0 2 5  
## very high 4 0 0 5 4  
##   
## $overall  
## Accuracy Kappa AccuracyLower AccuracyUpper AccuracyNull   
## 0.3174603 0.1505174 0.2058489 0.4468877 0.2857143   
## AccuracyPValue McnemarPValue   
## 0.3320083 NaN   
##   
## $byClass  
## Sensitivity Specificity Pos Pred Value Neg Pred Value  
## Class: very low 0.3888889 0.8000000 0.43750000 0.7659574  
## Class: low 0.4615385 0.9000000 0.54545455 0.8653846  
## Class: moderate 0.2500000 0.7627119 0.06666667 0.9375000  
## Class: high 0.1333333 0.8750000 0.25000000 0.7636364  
## Class: very high 0.3076923 0.8200000 0.30769231 0.8200000  
## Precision Recall F1 Prevalence Detection Rate  
## Class: very low 0.43750000 0.3888889 0.4117647 0.28571429 0.11111111  
## Class: low 0.54545455 0.4615385 0.5000000 0.20634921 0.09523810  
## Class: moderate 0.06666667 0.2500000 0.1052632 0.06349206 0.01587302  
## Class: high 0.25000000 0.1333333 0.1739130 0.23809524 0.03174603  
## Class: very high 0.30769231 0.3076923 0.3076923 0.20634921 0.06349206  
## Detection Prevalence Balanced Accuracy  
## Class: very low 0.2539683 0.5944444  
## Class: low 0.1746032 0.6807692  
## Class: moderate 0.2380952 0.5063559  
## Class: high 0.1269841 0.5041667  
## Class: very high 0.2063492 0.5638462  
##   
## $mode  
## [1] "sens\_spec"  
##   
## $dots  
## list()

# Combine the predictions with countries.  
country = data.frame(c\_test$location)  
cases = data.frame(knn\_pred\_cases)  
deaths = data.frame(knn\_pred\_deaths)  
covid19\_knn\_pred = cbind(country, cases, deaths)  
  
# Change column names  
labels = c("country", "cases", "deaths")  
colnames(covid19\_knn\_pred) = labels  
  
# Write prediction dataframe to csv file  
write.csv(covid19\_knn\_pred, 'knn\_pred.csv')  
  
## K-NN Analysis   
  
# For the infection rate, it looks like the accuracy of the model peaks around 19 neighboring points. For the death rate, the accuracy of the model peaks around 8 neightboring points. These values are used to optimize the model. The confusion matrix shows an Accuracy of between 40-50% which indicates that this model outperforms the decision tree and random forest models. The precision and recall have a range between 40-50% for both infection and death rate. From looking at the separate classes, the prediction fits the pattern shown from the previous models (more easily predictable extremities for infection rate and more easily predictable higher classifications for the death rate).  
  
# Remove unused columns  
c\_train$iso\_code = NULL  
c\_train$location = NULL  
  
# Get informatino Gain for infection  
infection\_weight = information.gain(total\_cases\_per\_million ~ . -total\_deaths\_per\_million, c\_train)  
infection\_weight

## attr\_importance  
## total\_deaths\_per\_million 0.5443465  
## total\_tests\_per\_thousand 0.0000000  
## population 0.0000000  
## population\_density 0.0000000  
## median\_age 0.2778736  
## aged\_65\_older 0.1949711  
## aged\_70\_older 0.2290800  
## gdp\_per\_capita 0.2593682  
## extreme\_poverty 0.1057138  
## cvd\_death\_rate 0.1609488  
## diabetes\_prevalence 0.0000000  
## female\_smokers 0.1469432  
## male\_smokers 0.0000000  
## handwashing\_facilities 0.1643933  
## hospital\_beds\_per\_100k 0.0000000

# Get informatino Gain for deaths  
death\_weight = information.gain(total\_deaths\_per\_million ~ . -total\_cases\_per\_million, c\_train)  
death\_weight

## attr\_importance  
## total\_cases\_per\_million 0.5443465  
## total\_tests\_per\_thousand 0.0000000  
## population 0.1128988  
## population\_density 0.0000000  
## median\_age 0.2013533  
## aged\_65\_older 0.2036021  
## aged\_70\_older 0.3099297  
## gdp\_per\_capita 0.2287570  
## extreme\_poverty 0.1022591  
## cvd\_death\_rate 0.1814535  
## diabetes\_prevalence 0.0000000  
## female\_smokers 0.1954103  
## male\_smokers 0.0000000  
## handwashing\_facilities 0.1386182  
## hospital\_beds\_per\_100k 0.0000000