INTRODUCTION TO DATA MINING

Isaiah Thompson Ocansey

2023-05-24

Introduction

Multi-class classification is a common problem in machine learning where the goal is to classify data instances into one of multiple predefined classes or categories. In this R Markdown document, we will explore the intricacies of multi-class classification problems using the R programming language. We will cover various aspects such as data exploration, model development, evaluation, and interpretation.

One of the initial steps we will consider in the in multi-class classification is data exploration. It involves understanding of the structure and characteristics of the weight lift data set we will be considering, identifying the classes and features, and gaining insights into their distributions and relationships.

Once we have gained a good understanding of the data, we will proceed to model development. R offers a wide range of algorithms for multi-class classification, including decision trees, random forests, support vector machines (SVM), logistic regression, neural networks, and more. These algorithms can be implemented using various packages, and we can tune their parameters to optimize their performance on the given dataset.

Evaluation is a critical step in multi-class classification, as it helps assess the performance of the models. R provides several evaluation metrics specifically designed for multi-class classification, such as accuracy, precision, recall, F1 score, and confusion matrices. We can use cross-validation techniques and resampling methods to obtain reliable estimates of model performance and compare different models effectively.

Interpreting the results of a multi-class classification model is essential for gaining insights and understanding the factors that contribute to the predictions. R Markdown allows us to generate interactive reports that combine code, visualizations, and text. We can use these reports to present and communicate our findings, making it easier to understand the model's behavior and the impact of different features on the classification results.

In this R Markdown document, we will explore practical examples and provide step-by-step guidance on how to perform multi-class classification using R. We will cover data preprocessing, model training, evaluation and result interpretation. By the end of this document.

Aim

The aim of this project is to analyze a data set obtained from a weight lifting exercise program, which involved six participants performing five different lifting techniques (A, B, C, D, and E) correctly and incorrectly, as indicated by the classe variable. Accelerometer data was collected from the belt, forearm, arm, and dumbbell of each participant. The primary goal is to apply machine learning algorithms to predict the classe variable.

Method

We shall employ four different machine learning methods such as the multinomial logistic regression, random forest,linear discriminant analysis (LDA) and artificial neural network to analyze the weight lift data set.

The best model based on prediction accuracy would be used for further predictions. We would first start of by loading and gaining insight into the data set.

```
data<-read.csv(file="WtLift.csv", header = T, sep=",")
dim(data)
```

```
## [1] 19442   160
```

The data set has 19442 observations and 160 variables with the classe variable being the dependent variable

```
sum(is.na(data))
```

```
## [1] 1907375
```

We have observed a total of 1907375 missing values in the data set so we will first find the missing rate, remove variables which have missing rates exceeding 60 percent and then impute the missing values.

```
missing_rate <- data.frame(stringsAsFactors = F)
nr <- NROW(data)
nc <- NCOL(data)
Var_name <- variable.names(data)
for (i in 1:nc) {
 na <- sum(is.na(data[,i]))
 na_rate <- (na/nr)*100
 result <- list(Variable = Var_name[i], Number_Missing = na,
 Missing_Rate = na_rate)
 missing_rate <- rbind(missing_rate, result, stringsAsFactors = F)
}
```

The code above computes the proportion of missing values of each variable in the data set

After computing the missing rate, we shall now proceed to remove variables with missing rate exceeding 60 percent using the following codes.

```
removemissing <- missing_rate$Missing_Rate > 60.0
removemissing1 <- missing_rate$Missing_Rate == 0.0
No.missing <- names(data[, removemissing1])
sum(!removemissing)
```

```
## [1] 60
```

```
data <- data[, !removemissing]
dim(data)
```

```
## [1] 19442    60
```

The code above checks for missing rates of variables above 60 percent and removes them from the data set. After removing variables with missing rates above 60 percent, we now have 19442 observations and 60 variables to impute

```
suppressPackageStartupMessages(library(mice))
data_imputed <- mice(data[,-1],maxit=15, method='cart', seed=125)
dat <- complete(data_imputed, 1)
dat <- as.data.frame(dat)
rm(data_imputed);dim(dat)
```

The code above imputes all missing values of the remaining variables. After missing value imputation, we shall now confirm if indeed, there are no missing values in the remaining data set

```
table(is.na(dat))
```

```
##
##   FALSE
## 1147078
```

The above codes confirms there are no more missing values in our new data set labeled dat

Exploratory Data Analysis

We shall now gain insight into the data set dat to better understand the variables in the data by way of numerical and graphical exploratory data analysis.

```
freq(dat$classe,total = T)
```

```
##           n     %  val%
## A      5532  28.5  28.5
## B      3766  19.4  19.4
## C      3383  17.4  17.4
## D      3190  16.4  16.4
## E      3571  18.4  18.4
## Total 19442 100.0 100.0
```

The target variable classe has five distinct classes; A, B,C,D and E. The above code tabulates the number of observations in each distinct class, It can be observed from the above frequency table that classes; A,B,C,D and E have 5532,3766,3383,3190 and 3571 observations representing 28.5,19.4,17.4, 16.4 and 18.4 percent respectively.

We shall now check the type of variables we are working with

```
str(dat)
```

```
## 'data.frame':    19442 obs. of  59 variables:
##  $ user_name           : chr  "carlitos" "carlitos" "carlitos" "carlitos" ...
##  $ raw_timestamp_part_1: int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 1323
##  $ raw_timestamp_part_2: int  788290 808298 820366 120339 196328 304277 368296 440390 484323 484434
##  $ cvtd_timestamp      : chr  "05/12/2011 11:23" "05/12/2011 11:23" "05/12/2011 11:23" "05/12/2011 1
##  $ new_window          : chr  "no" "no" "no" "no" ...
##  $ num_window          : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt           : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt          : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
```

```
##  $ yaw_belt            : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
##  $ total_accel_belt    : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ gyros_belt_x        : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03 ...
##  $ gyros_belt_y        : num  0 0 0 0 0.02 0 0 0 0 0 ...
##  $ gyros_belt_z        : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0 ...
##  $ accel_belt_x        : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y        : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z        : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ magnet_belt_x       : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y       : int  599 608 600 604 600 603 599 603 602 609 ...
##  $ magnet_belt_z       : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308 ...
##  $ roll_arm            : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm           : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6 ...
##  $ yaw_arm             : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm     : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ gyros_arm_x         : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02 ...
##  $ gyros_arm_y         : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 ...
##  $ gyros_arm_z         : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02 ...
##  $ accel_arm_x         : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y         : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z         : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ magnet_arm_x        : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_y        : int  337 337 344 344 337 342 336 338 341 334 ...
##  $ magnet_arm_z        : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ roll_dumbbell       : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell      : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell        : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ total_accel_dumbbell: int  37 37 37 37 37 37 37 37 37 37 ...
##  $ gyros_dumbbell_x    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ gyros_dumbbell_y    : num  -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 ...
##  $ gyros_dumbbell_z    : num  0 0 0 -0.02 0 0 0 0 0 0 ...
##  $ accel_dumbbell_x    : int  -234 -233 -232 -232 -233 -234 -232 -234 -232 -235 ...
##  $ accel_dumbbell_y    : int  47 47 46 48 48 48 47 46 47 48 ...
##  $ accel_dumbbell_z    : int  -271 -269 -270 -269 -270 -269 -270 -272 -269 -270 ...
##  $ magnet_dumbbell_x   : int  -559 -555 -561 -552 -554 -558 -551 -555 -549 -558 ...
##  $ magnet_dumbbell_y   : int  293 296 298 303 292 294 295 300 292 291 ...
##  $ magnet_dumbbell_z   : num  -65 -64 -63 -60 -68 -66 -70 -74 -65 -69 ...
##  $ roll_forearm        : num  28.4 28.3 28.3 28.1 28 27.9 27.9 27.8 27.7 27.7 ...
##  $ pitch_forearm       : num  -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.8 -63.8 -63.8 ...
##  $ yaw_forearm         : num  -153 -153 -152 -152 -152 -152 -152 -152 -152 -152 ...
##  $ total_accel_forearm : int  36 36 36 36 36 36 36 36 36 36 ...
##  $ gyros_forearm_x     : num  0.03 0.02 0.03 0.02 0.02 0.02 0.02 0.02 0.03 0.02 ...
##  $ gyros_forearm_y     : num  0 0 -0.02 -0.02 0 -0.02 0 -0.02 0 0 ...
##  $ gyros_forearm_z     : num  -0.02 -0.02 0 0 -0.02 -0.03 -0.02 0 -0.02 -0.02 ...
##  $ accel_forearm_x     : int  192 192 196 189 189 193 195 193 193 190 ...
##  $ accel_forearm_y     : int  203 203 204 206 206 203 205 205 204 205 ...
##  $ accel_forearm_z     : int  -215 -216 -213 -214 -214 -215 -215 -213 -214 -215 ...
##  $ magnet_forearm_x    : int  -17 -18 -18 -16 -17 -9 -18 -9 -16 -22 ...
##  $ magnet_forearm_y    : num  654 661 658 658 655 660 659 660 653 656 ...
##  $ magnet_forearm_z    : num  476 473 469 469 473 478 470 474 476 473 ...
##  $ classe              : chr  "A" "A" "A" "A" ...
```

It can be observed from the above output that apart from the target variable, classe, the variables; user_name, cvtd_timestamp and new_window are character variables while all other variables are either discrete or

continuous.

```
#ggplot(data = dat) +
#labs(title = "Distribution of classe Variable")
  #geom_bar(mapping = aes(x = classe))
```

The above bar chart shows the frequency of each distinct class of the classe variable. This has been muted due to knitting to pdf issues. It can be observed that class A has higher frequency followed by B and E.

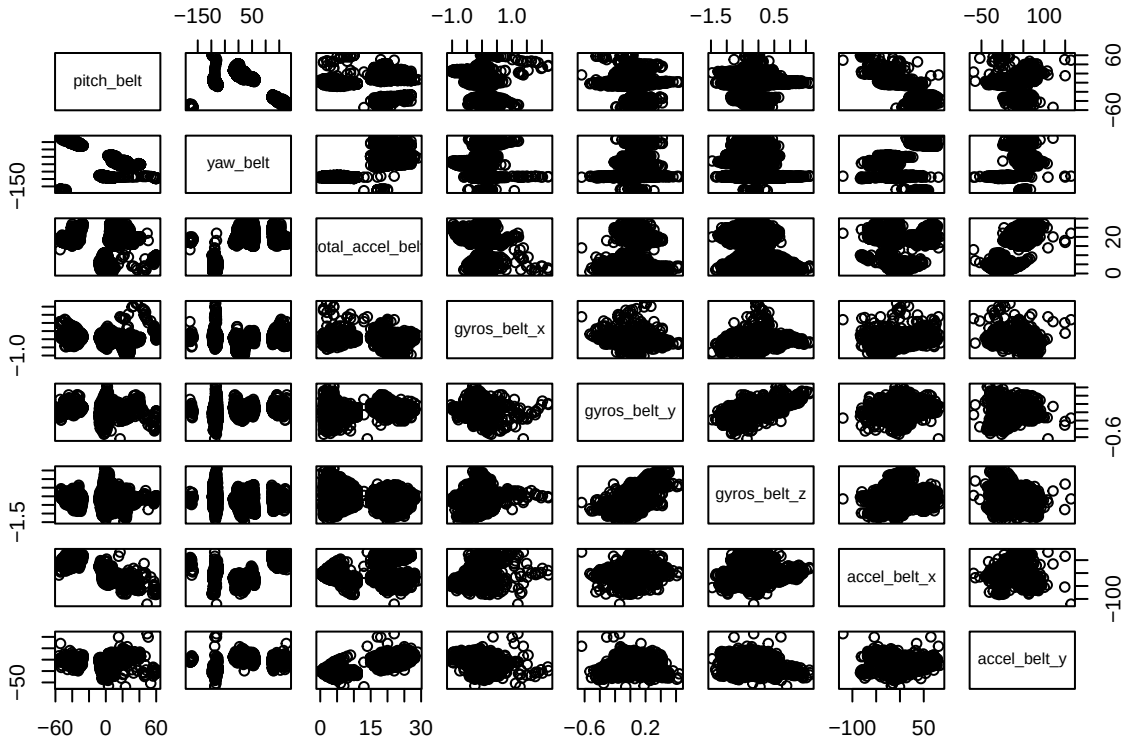
We chose randomly pitch_arm and pitch_forearm to visualize their distributions

```
#plot(density(data$pitch_arm))
#ggplot(data, aes(x = pitch_forearm)) +
#  geom_density()
```

It can be observed that none of the chosen variables are normally distributed

Also, we randomly chose pitch_belt,yaw_belt,total_accel_belt, gyros x,yz, accel_belt_x,y to visualize their pairwise correlation.

```
pairs(dat[,c(8:15)])
```



It can be observed from the above that some of the variables are somewhat positively correlated but overall, this is negligible.

Data Partition

For the purpose of training our models, we split the data set into training and test in the ratio 2:1 respectively.

```
set.seed(123)
sample <- sample(nrow(dat), (2/3)*nrow(dat), replace = FALSE)
# training set
training <- dat[sample, ]
#test set
test <- dat[-sample, ]
yobs <- test$classe
dim(training); dim(test)
```

```
## [1] 12961    59
```

```
## [1] 6481   59
```

After partitioning the data set, the training set has 12961 observations and 59 variables while the test set has 6481 observations and 59 variables.

Base Model: Multinomial Logistic Regression Model

We will now proceed to training the multinomial logistic model as our base model.

Briefly, multinomial Logistic Regression is a statistical modeling technique used to predict categorical outcomes with three or more unordered response categories. This report uses the Multinomial Logistic Regression model to predict the outcome of the classe variable in the dat data set.

```
# Define the model
fit.mlr <- train(classe ~ ., data = training, method = "multinom", trControl = trainControl(method = "cv
```

```
## # weights:  410 (324 variable)
## initial  value 16688.261714
## iter  10 value 15708.483614
## iter  20 value 12528.836589
## iter  30 value 11957.173283
## iter  40 value 11579.475785
## iter  50 value 10320.248974
## iter  60 value 10072.035455
## iter  70 value 9939.444789
## iter  80 value 9863.294762
## iter  90 value 9807.475407
## iter 100 value 9658.369986
## final  value 9658.369986
## stopped after 100 iterations
## # weights:  410 (324 variable)
## initial  value 16688.261714
## iter  10 value 15708.483614
## iter  20 value 12528.836679
## iter  30 value 11957.173484
## iter  40 value 11579.476146
```

```
## iter  50 value 10320.168791
## iter  60 value 10070.581307
## iter  70 value 9931.891377
## iter  80 value 9796.224095
## iter  90 value 9775.337179
## iter  90 value 9775.337178
## final  value 9775.090068
## converged
## # weights:  410 (324 variable)
## initial  value 16688.261714
## iter  10 value 15708.483614
## iter  20 value 12528.836589
## iter  30 value 11957.173283
## iter  40 value 11579.475785
## iter  50 value 10319.994779
## iter  60 value 10071.062182
## iter  70 value 9935.948853
## iter  80 value 9864.958146
## iter  90 value 9815.126590
## iter 100 value 9708.574692
## final  value 9708.574692
## stopped after 100 iterations
## # weights:  410 (324 variable)
## initial  value 16688.261714
## iter  10 value 15509.097037
## iter  20 value 11945.353127
## iter  30 value 11248.369203
## iter  40 value 10582.835430
## iter  50 value 10129.076900
## iter  60 value 9727.858478
## iter  70 value 9633.313010
## iter  80 value 9491.686960
## iter  90 value 9401.164401
## iter 100 value 9342.546559
## final  value 9342.546559
## stopped after 100 iterations
## # weights:  410 (324 variable)
## initial  value 16688.261714
## iter  10 value 15509.097039
## iter  20 value 11945.353215
## iter  30 value 11248.369506
## iter  40 value 10582.836173
## iter  50 value 10129.902182
## iter  60 value 9741.389410
## iter  70 value 9641.449671
## iter  80 value 9504.166938
## iter  90 value 9405.530254
## iter 100 value 9328.131593
## final  value 9328.131593
## stopped after 100 iterations
## # weights:  410 (324 variable)
## initial  value 16688.261714
## iter  10 value 15509.097037
## iter  20 value 11945.353127
```

```
## iter  30 value 11248.369203
## iter  40 value 10582.835430
## iter  50 value 10130.212007
## iter  60 value 9748.052416
## iter  70 value 9681.063813
## iter  80 value 9491.021071
## iter  90 value 9368.239238
## iter 100 value 9322.633112
## final  value 9322.633112
## stopped after 100 iterations
## # weights:  410 (324 variable)
## initial  value 16688.261714
## iter  10 value 15688.371822
## iter  20 value 12186.789894
## iter  30 value 11470.661396
## iter  40 value 10839.130650
## iter  50 value 10744.712416
## iter  60 value 9854.323561
## iter  70 value 9755.989365
## iter  80 value 9472.510669
## iter  90 value 9385.795958
## iter 100 value 9352.255677
## final  value 9352.255677
## stopped after 100 iterations
## # weights:  410 (324 variable)
## initial  value 16688.261714
## iter  10 value 15688.371824
## iter  20 value 12186.789981
## iter  30 value 11470.661649
## iter  40 value 10839.131516
## iter  50 value 10744.713383
## iter  60 value 9853.680930
## iter  70 value 9754.256338
## iter  80 value 9467.153973
## iter  90 value 9376.716754
## iter 100 value 9300.948463
## final  value 9300.948463
## stopped after 100 iterations
## # weights:  410 (324 variable)
## initial  value 16688.261714
## iter  10 value 15688.371822
## iter  20 value 12186.789894
## iter  30 value 11470.661396
## iter  40 value 10839.130651
## iter  50 value 10744.712416
## iter  60 value 9850.950162
## iter  70 value 9743.729128
## iter  80 value 9464.162129
## iter  90 value 9361.696053
## iter 100 value 9314.795072
## final  value 9314.795072
## stopped after 100 iterations
## # weights:  410 (324 variable)
## initial  value 16688.261714
```

```
## iter  10 value 15364.031439
## iter  20 value 11998.327793
## iter  30 value 11261.394386
## iter  40 value 10618.751096
## iter  50 value 10285.546019
## iter  60 value 9856.817362
## iter  70 value 9764.626566
## iter  80 value 9611.535068
## iter  90 value 9444.653724
## iter 100 value 9146.905150
## final  value 9146.905150
## stopped after 100 iterations
## # weights:  410 (324 variable)
## initial  value 16688.261714
## iter  10 value 15364.031440
## iter  20 value 11998.327877
## iter  30 value 11261.394638
## iter  40 value 10618.751831
## iter  50 value 10285.562301
## iter  60 value 9863.289378
## iter  70 value 9787.573506
## iter  80 value 9424.950181
## iter  90 value 9312.232271
## iter 100 value 9263.372526
## final  value 9263.372526
## stopped after 100 iterations
## # weights:  410 (324 variable)
## initial  value 16688.261714
## iter  10 value 15364.031439
## iter  20 value 11998.327793
## iter  30 value 11261.394386
## iter  40 value 10618.751097
## iter  50 value 10285.579664
## iter  60 value 9872.073841
## iter  70 value 9805.792585
## iter  80 value 9448.955789
## iter  90 value 9393.106334
## iter 100 value 9319.523305
## final  value 9319.523305
## stopped after 100 iterations
## # weights:  410 (324 variable)
## initial  value 16686.652276
## iter  10 value 15363.273807
## iter  20 value 12429.285514
## iter  30 value 11834.767732
## iter  40 value 11450.594950
## iter  50 value 10163.123904
## iter  60 value 10002.566288
## iter  70 value 9883.123254
## iter  80 value 9781.028194
## iter  90 value 9679.056126
## iter 100 value 9628.004339
## final  value 9628.004339
## stopped after 100 iterations
```

```
## # weights:  410 (324 variable)
## initial  value 16686.652276
## iter  10 value 15363.273808
## iter  20 value 12429.285627
## iter  30 value 11834.767993
## iter  40 value 11450.595477
## iter  50 value 10163.823025
## iter  60 value 10013.113289
## iter  70 value 9904.965022
## iter  80 value 9900.692769
## iter  80 value 9900.692768
## final  value 9900.666450
## converged
## # weights:  410 (324 variable)
## initial  value 16686.652276
## iter  10 value 15363.273807
## iter  20 value 12429.285514
## iter  30 value 11834.767732
## iter  40 value 11450.594951
## iter  50 value 10164.045166
## iter  60 value 10015.994421
## iter  70 value 9940.153618
## iter  80 value 9823.009507
## iter  90 value 9763.189163
## iter 100 value 9760.895032
## final  value 9760.895032
## stopped after 100 iterations
## # weights:  410 (324 variable)
## initial  value 20859.924783
## iter  10 value 19350.706936
## iter  20 value 15084.322352
## iter  30 value 14191.631076
## iter  40 value 13614.602245
## iter  40 value 13614.602245
## iter  50 value 12673.632012
## iter  60 value 12072.405980
## iter  70 value 11961.777625
## iter  80 value 11714.194347
## iter  90 value 11629.412187
## iter 100 value 11600.077392
## final  value 11600.077392
## stopped after 100 iterations
```

```
# Print the cross-validation results
print(fit.mlr$results)
```

```
##   decay  Accuracy     Kappa AccuracySD    KappaSD
## 1 0e+00 0.6565860 0.5641030 0.01311126 0.01634268
## 2 1e-04 0.6570487 0.5645890 0.01044991 0.01327654
## 3 1e-01 0.6602124 0.5686742 0.01498529 0.01934408
```

The code above performs 5-fold cross-validation (number = 5) and outputs the cross-validation results, including accuracy, Kappa statistic, and other performance metrics

Model Evaluation of the Multinomial Binomial Logistic Regression

```
pred.mlr <- predict(fit.mlr, newdata = test)
```

The predictions based on the multinomial logistic model has not been output in order not to make the work cumbersome. We are interested in the accuarcy of the predictions

```
# Compute overall accuracy
accuracy.mlr <- mean(pred.mlr == yobs)
accuracy.mlr
```

```
## [1] 0.6653294
```

The model accuracy of the multinomial logistic regression is 66.05 percent meaning that 66.05 percent of the time, it is able to accurately classify the distinct classes of the response variable, classe

Random Forest Model

We will now proceed to train the Random Forest Model.

Briefly, Random Forest is a popular algorithm for multiclass classification tasks due to its ability to handle complex data sets, capture nonlinear relationships, and provide robust predictions.

In this approach, multiple decision trees are trained on subsets of the data set, where each tree is built using a random selection of features and observations. During the training process, each tree in the Random Forest independently makes predictions for a given instance based on its set of randomly selected features. The final prediction is determined by aggregating the predictions of all the trees in the forest through voting or averaging.

```
fit.rf <- randomForest(factor(classe) ~., data=training,importance=TRUE, proximity=TRUE, ntree=500)
fit.rf
```

```
##
## Call:
##  randomForest(formula = factor(classe) ~ ., data = training, importance = TRUE,      proximity = TRU
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.15%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3667    1    0    0    0 0.0002726281
## B    3 2527    0    0    0 0.0011857708
## C    0    2 2283    3    0 0.0021853147
## D    0    0    5 2074    2 0.0033637674
## E    0    0    0    3 2391 0.0012531328
```

The RF model above has done well on the training by almost correctly classifying each distinct classes of the classe variable and we shall not further fine tune it's parameters,

We shall now proceed to see which variables are more relevant in the RF Model.

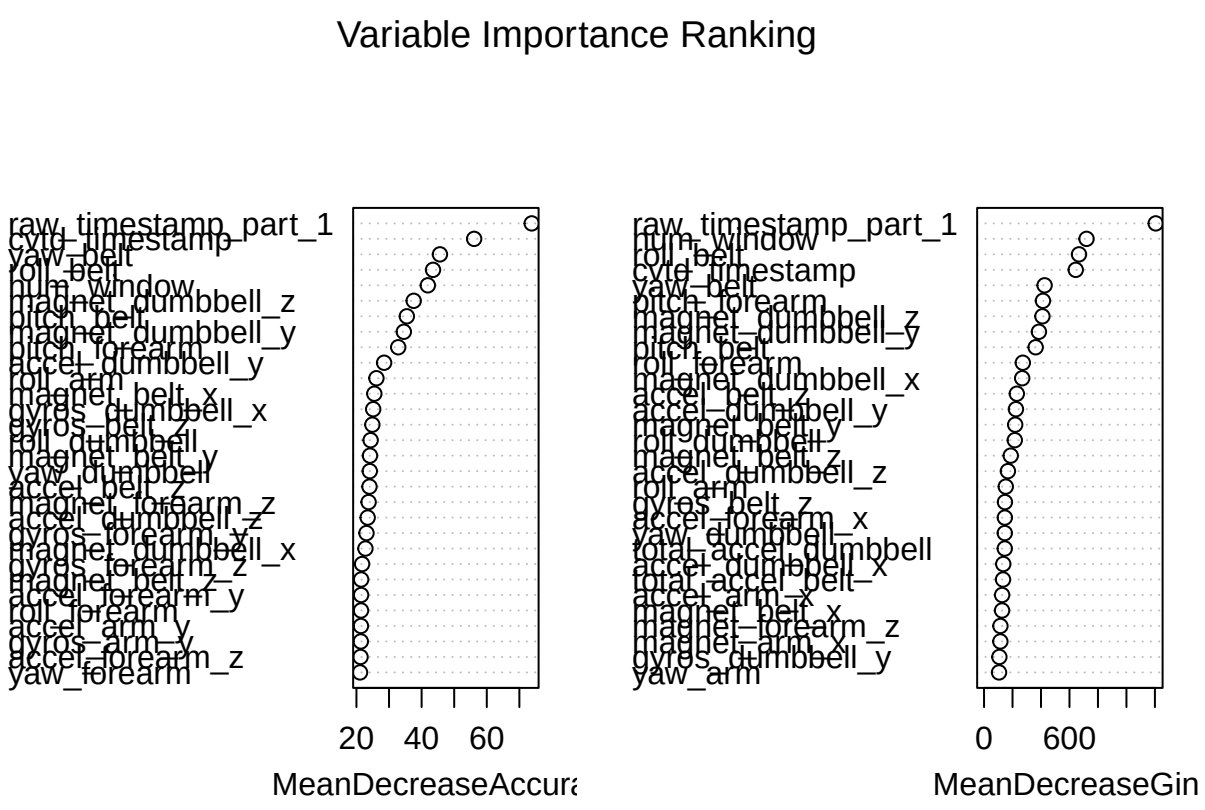
```
# VARIABLE IMPORTANCE RANKING
round(importance(fit.rf), 2);
```

```
##                            A     B     C     D     E MeanDecreaseAccuracy
## user_name              10.92 10.98  9.33 10.42 10.93                 12.86
## raw_timestamp_part_1   49.08 55.15 55.03 56.73 41.30                 73.79
## raw_timestamp_part_2    6.19  9.45  9.81  9.07  7.74                 17.83
## cvtd_timestamp         34.07 42.54 37.81 44.40 46.89                 56.12
## new_window              0.02 -0.99 -0.46 -1.00  0.45                 -0.96
## num_window             28.85 38.09 41.44 34.05 33.77                 41.91
## roll_belt              28.42 38.57 34.42 39.19 35.92                 43.50
## pitch_belt             22.69 34.35 28.34 27.79 23.61                 35.47
## yaw_belt               28.55 32.73 31.51 38.25 24.18                 45.62
## total_accel_belt       12.09 13.18 11.63 13.42 13.50                 14.44
## gyros_belt_x           12.51 12.42 15.70 12.04 12.04                 18.41
## gyros_belt_y            9.44 12.10 12.93 12.30 12.57                 15.35
## gyros_belt_z           16.28 21.41 20.19 19.70 20.32                 24.88
## accel_belt_x           11.00 13.37 13.96 11.61 10.27                 15.20
## accel_belt_y           10.51 12.56 11.42 14.73 10.94                 14.89
## accel_belt_z           19.53 21.29 18.81 20.12 17.39                 24.00
## magnet_belt_x          13.57 21.56 19.53 19.77 18.82                 25.48
## magnet_belt_y          17.24 22.41 18.89 22.96 20.67                 24.16
## magnet_belt_z          15.87 19.39 17.62 22.28 17.21                 21.44
## roll_arm               15.51 23.23 20.58 22.40 17.38                 26.13
## pitch_arm              12.39 19.21 16.47 16.74 12.99                 19.98
## yaw_arm                15.57 18.43 17.65 19.29 14.35                 20.91
## total_accel_arm         7.48 15.70 12.21 11.48 11.62                 15.42
## gyros_arm_x            11.09 12.88 14.25 12.54 10.53                 15.11
## gyros_arm_y            12.22 17.03 15.16 16.88  9.67                 21.34
## gyros_arm_z             9.73 11.24 11.46 10.72  7.48                 18.79
## accel_arm_x            14.16 15.27 15.90 16.43 13.82                 16.51
## accel_arm_y            13.17 16.95 13.93 14.40 12.84                 21.37
## accel_arm_z            10.99 14.53 14.98 15.24 12.66                 16.78
## magnet_arm_x           12.94 13.29 14.22 14.00 12.37                 14.36
## magnet_arm_y           10.51 15.37 14.83 15.52 11.15                 15.26
## magnet_arm_z           13.61 20.47 15.65 13.80 12.70                 20.47
## roll_dumbbell          17.41 21.77 23.59 22.56 20.16                 24.35
## pitch_dumbbell          9.46 16.27 15.28 12.44 12.14                 14.85
## yaw_dumbbell           15.96 19.93 21.53 18.31 18.06                 24.09
## total_accel_dumbbell   15.44 18.39 16.61 17.98 17.21                 20.69
## gyros_dumbbell_x       11.89 17.71 17.21 16.97 15.11                 25.14
## gyros_dumbbell_y       15.45 15.78 21.33 16.46 13.42                 19.41
## gyros_dumbbell_z       10.44 15.79 12.82 11.70 10.80                 19.83
## accel_dumbbell_x       13.08 19.00 17.58 15.12 16.19                 19.29
## accel_dumbbell_y       21.04 22.84 25.59 23.53 20.96                 28.49
## accel_dumbbell_z       15.34 20.89 21.30 19.26 18.53                 23.44
## magnet_dumbbell_x      19.41 21.54 24.11 20.79 18.73                 22.76
## magnet_dumbbell_y      28.22 30.41 34.93 28.98 25.91                 34.54
## magnet_dumbbell_z      35.09 31.00 36.31 27.51 26.68                 37.57
## roll_forearm           22.05 18.98 23.05 16.67 16.55                 21.41
## pitch_forearm          23.36 27.87 31.06 29.32 25.30                 32.82
## yaw_forearm            12.12 15.19 15.48 17.03 15.79                 21.10
## total_accel_forearm    13.65 14.48 14.43 10.29 11.95                 16.90
```

```
## gyros_forearm_x       7.59 11.92 13.31 11.44 10.45                20.13
## gyros_forearm_y      10.38 20.74 18.03 16.89 13.69                23.08
## gyros_forearm_z       9.15 17.14 13.33 10.75 12.19                21.73
## accel_forearm_x      13.16 19.50 18.08 20.85 17.87                19.98
## accel_forearm_y      13.77 16.53 18.15 13.74 11.92                21.43
## accel_forearm_z      14.10 18.73 17.13 16.24 14.38                21.26
## magnet_forearm_x     11.78 16.55 14.79 14.39 14.34                17.11
## magnet_forearm_y     13.60 16.66 16.62 17.41 15.00                20.32
## magnet_forearm_z     15.81 20.48 17.42 18.89 16.49                23.72
##                     MeanDecreaseGini
## user_name                      50.86
## raw_timestamp_part_1         1205.72
## raw_timestamp_part_2           12.79
## cvtd_timestamp                643.98
## new_window                      0.36
## num_window                    719.88
## roll_belt                     666.90
## pitch_belt                    362.62
## yaw_belt                      425.38
## total_accel_belt              133.35
## gyros_belt_x                   46.79
## gyros_belt_y                   58.00
## gyros_belt_z                  146.98
## accel_belt_x                   65.11
## accel_belt_y                   78.80
## accel_belt_z                  230.30
## magnet_belt_x                 126.67
## magnet_belt_y                 218.81
## magnet_belt_z                 187.60
## roll_arm                      152.62
## pitch_arm                      74.76
## yaw_arm                       105.29
## total_accel_arm                35.47
## gyros_arm_x                    47.23
## gyros_arm_y                    49.10
## gyros_arm_z                    22.02
## accel_arm_x                   127.05
## accel_arm_y                    63.98
## accel_arm_z                    50.75
## magnet_arm_x                  114.68
## magnet_arm_y                   88.08
## magnet_arm_z                   66.82
## roll_dumbbell                 216.01
## pitch_dumbbell                 93.64
## yaw_dumbbell                  145.72
## total_accel_dumbbell          145.63
## gyros_dumbbell_x               52.45
## gyros_dumbbell_y              107.75
## gyros_dumbbell_z               30.81
## accel_dumbbell_x              135.19
## accel_dumbbell_y              224.02
## accel_dumbbell_z              167.61
## magnet_dumbbell_x             268.09
## magnet_dumbbell_y             385.39
```

```
## magnet_dumbbell_z               409.96
## roll_forearm                    272.28
## pitch_forearm                   413.78
## yaw_forearm                      69.00
## total_accel_forearm              42.55
## gyros_forearm_x                  28.32
## gyros_forearm_y                  44.66
## gyros_forearm_z                  29.83
## accel_forearm_x                 146.07
## accel_forearm_y                  54.73
## accel_forearm_z                 101.31
## magnet_forearm_x                 86.50
## magnet_forearm_y                 83.56
## magnet_forearm_z                115.02
```

```
varImpPlot(fit.rf, main="Variable Importance Ranking")
```



It can be observed that raw_timestam_part_1,cvd_timestand,num_window, yaw_belt and the magnet_dumbbell_z are the five most relevant variables in the RF model in this instance.

Model Evaluation of the Random Forest Model

```
pred.rf <- predict(fit.rf, newdata=test)
```

```
test$pred.rf= pred.rf
cft<-table(test$classe,test$pred.rf)

cft
```

```
##
##        A    B    C    D    E
##   A 1864    0    0    0    0
##   B    1 1235    0    0    0
##   C    0    3 1092    0    0
##   D    0    0    0 1109    0
##   E    0    0    0    1 1176
```

It can be observed from the confusion table above that the RF model correctly classify the distinct classes of the classe variable except one

```
accuracy.rf<-mean(test$classe==test$pred.rf)
accuracy.rf
```

```
## [1] 0.9992285
```

The RF model has 99.89 percent accuracy which means it is able to correctly classify the distinct classes of the classe variable 99.89 of time.

Linear Discriminant Analysis (LDA)

We shall ow proceed to train the Linear Discriminant Analysis model (LDA).

Briefly, LDA is a dimensionality reduction and classification technique used to find a linear combination of features that maximizes class separation in a supervised learning setting. The main objective of LDA is to project the original feature space onto a lower-dimensional space while maximizing the separation between different classes. LDA accomplishes this by finding a set of discriminant functions that maximize the ratio of between-class scatter to within-class scatter. We shall use this model to predict the classification of the classe variable that we are considering.

```
fit.lda <- lda(classe ~ ., data=training, CV=F)
```

The above fits the LDA model using the training data set

Model Evaluation of LDA

We proceed to predict using the test data set

```
pred.lda<- predict(fit.lda, newdata=test)
pred_class<-pred.lda$class
table(pred_class,test$classe)
```

```
##
## pred_class    A    B    C    D    E
##          A 1683  141    1    0    0
##          B  152  916   87    1    0
##          C   28  170  977  126    5
##          D    0    9   30  931  107
##          E    1    0    0   51 1065
```

From the confusion table above, it can be observed that the LDA model also did quite well in classifying the distinct classes of the classe variable but has error in the classification

```
accuracy.lda<-mean(pred_class==test$classe)
accuracy.lda
```

```
## [1] 0.8597439
```

We can observe that the LDA model has 85.97 percent accuracy meaning that 85.97 of the times, it is able to correctly classify the distinct classes of the classe variable. ‘

Artificial Neural Network

We shall now train our last model which is artificial neural network. Briefly, Artificial Neural Networks (ANNs) consist of interconnected nodes, known as artificial neurons or “units,” organized into layers.

We shall use ANN to classify the different classes of the classe variable based on patterns and relationships learned from the training data.

```
training$classe <- as.factor(training$classe)
nnclas_model <- nnet(classe ~ ., data = training[,-c(1,4,5)], size = 5, decay = 0.0001,maxit = 500)
```

```
## # weights:  310
## initial  value 22043.857465
## final  value 20581.208950
## converged
```

The above output shows the fit for the artificial neural network model with 500 maximum iterations and 4 units in the hidden layer

Model Evaluation of the Artificial Neural Network

We proceed to predict using the test data

```
yhat <- predict(nnclas_model, test[,-c(1,4,5)], type = 'class')
confusionMatrix(as.factor(yhat),as.factor(test$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
```

```
##          A 1864 1236 1095 1109 1177
##          B    0    0    0    0    0
##          C    0    0    0    0    0
##          D    0    0    0    0    0
##          E    0    0    0    0    0
##
## Overall Statistics
##
##                Accuracy : 0.2876
##                  95% CI : (0.2766, 0.2988)
##     No Information Rate : 0.2876
##     P-Value [Acc > NIR] : 0.5047
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.0000    0.000   0.0000   0.0000
## Specificity            0.0000   1.0000    1.000   1.0000   1.0000
## Pos Pred Value         0.2876      NaN      NaN      NaN      NaN
## Neg Pred Value            NaN   0.8093    0.831   0.8289   0.8184
## Prevalence             0.2876   0.1907    0.169   0.1711   0.1816
## Detection Rate         0.2876   0.0000    0.000   0.0000   0.0000
## Detection Prevalence   1.0000   0.0000    0.000   0.0000   0.0000
## Balanced Accuracy      0.5000   0.5000    0.500   0.5000   0.5000
```

```
accuracy.nn<-0.2876
accuracy.nn
```

```
## [1] 0.2876
```

The artificial neural network model performed poorly in classifying the distinct classes of the classe variable since it has model accuracy of 28.76 percent meaning that it is able to correctly classify the classe variable 28.76 of the time.

Discussion

We will now compare the performance of the four classification models we have used to see which is the best based on accuracy and we shall use the best model for further predictions.

Model Comparism

```
Measure <- c(accuracy.mlr,accuracy.rf,accuracy.lda,accuracy.nn)
Measures <- data.frame("Method"= c( "Multinomial Logistic Regression",   "Random Forest Model","Linear
knitr::kable(Measures, align = "lc")
```

Method	Prediction.Accuracy.Measures
Multinomial Logistic Regression	0.6653294
Random Forest Model	0.9992285
Linear Discriminat Analysis	0.8597439
Artificial Neural Network	0.2876000

It can be observed from the table above that the Random Forest model performed best followed by the Linear Discriminant Analysis model, Multinomial Logistic Regression model and then the Artificial Neural Network Model. We shall now use the Random Forest Model for further predictions.

Prediction with Best Model

Cleaning Test Data

```
new_dat<-read.csv(file="test.csv", header=T, sep = ",")
 dim(new_dat)
```

```
## [1] 180 159
```

The test data has 180 observations and 159 variables The test data has missing values so we went to clean the data just as we did for the weight lift data set

** Missing Value Imputation of Test Data**

```
missing_rate <- data.frame(stringsAsFactors = F)
nr <- NROW(new_dat)
nc <- NCOL(new_dat)
Var_name <- variable.names(new_dat)
for (i in 1:nc) {
 na <- sum(is.na(new_dat[,i]))
 na_rate <- (na/nr)*100
 result <- list(Variable = Var_name[i], Number_Missing = na,
 Missing_Rate = na_rate)
 missing_rate <- rbind(missing_rate, result, stringsAsFactors = F)
}
```

The above shows the missing rate of each variable in the data set

```
# removing variables with missing percentages > 60%
removemissing <- missing_rate$Missing_Rate > 60.0
removemissing1 <- missing_rate$Missing_Rate == 0.0
No.missing <- names(new_dat[, removemissing1])
sum(!removemissing)
```

```
## [1] 59
```

```
dat0 <- new_dat[, !removemissing]
dim(dat0)
```

```
## [1] 180  59
```

in the above output,We remove variables with missing rate above 60 percent

```
suppressPackageStartupMessages(library(mice))
data_imputed0 <- mice(dat0[,-1],maxit=15, method='cart', seed=125)
```

```
##
##  iter imp variable
##   1   1
##   1   2
##   1   3
##   1   4
##   1   5
##   2   1
##   2   2
##   2   3
##   2   4
##   2   5
##   3   1
##   3   2
##   3   3
##   3   4
##   3   5
##   4   1
##   4   2
##   4   3
##   4   4
##   4   5
##   5   1
##   5   2
##   5   3
##   5   4
##   5   5
##   6   1
##   6   2
##   6   3
##   6   4
##   6   5
##   7   1
##   7   2
##   7   3
##   7   4
##   7   5
##   8   1
##   8   2
##   8   3
##   8   4
##   8   5
##   9   1
##   9   2
##   9   3
##   9   4
```

```
##   9   5
##   10   1
##   10   2
##   10   3
##   10   4
##   10   5
##   11   1
##   11   2
##   11   3
##   11   4
##   11   5
##   12   1
##   12   2
##   12   3
##   12   4
##   12   5
##   13   1
##   13   2
##   13   3
##   13   4
##   13   5
##   14   1
##   14   2
##   14   3
##   14   4
##   14   5
##   15   1
##   15   2
##   15   3
##   15   4
##   15   5
```

```
dat1 <- complete(data_imputed0, 1)
dat1 <- as.data.frame(dat1)
rm(data_imputed0)
dim(dat1)
```

```
## [1] 180  58
```

we impute the missing variables using the mice package

Confirming Test Data has no Missing Value

```
sum(is.na(dat1))
```

```
## [1] 0
```

Prediction with Test Data

```
prediction1<-predict(fit.rf, newdata=dat1)
prediction2 <- predict(fit.rf, newdata=dat1, type = "prob")
table(prediction1)
```

```
## prediction1
##  A  B  C  D  E
## 48 31 40 26 35
```

```
prediction1;prediction2
```

```
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
##   B   E   E   E   D   B   C   A   E   A   B   E   E   B   C   A   D   A   A   B
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
##   B   C   C   B   A   A   D   C   B   C   B   B   E   C   D   C   A   A   A   C
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
##   A   A   A   C   A   E   C   C   D   D   B   B   E   A   A   D   A   C   C   C
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
##   D   B   A   D   E   A   E   A   B   A   C   B   D   E   C   D   E   E   B   C
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
##   D   D   B   E   D   E   B   C   A   A   C   E   C   E   E   C   A   D   C   A
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
##   E   D   A   B   B   A   C   E   C   C   B   D   E   A   D   B   A   B   D   A
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
##   D   D   E   A   C   E   C   C   E   E   B   B   B   A   A   A   A   A   D   E
## 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
##   A   B   B   B   C   A   C   D   E   D   A   E   C   E   E   E   A   A   A   A
## 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
##   A   A   B   C   C   C   D   C   A   E   C   C   A   A   C   E   E   D   C   B
## Levels: A B C D E
```

```
##         A     B     C     D     E
## 1   0.036 0.918 0.016 0.014 0.016
## 2   0.002 0.016 0.004 0.000 0.978
## 3   0.016 0.080 0.008 0.012 0.884
## 4   0.012 0.028 0.036 0.014 0.910
## 5   0.048 0.126 0.216 0.600 0.010
## 6   0.038 0.854 0.072 0.004 0.032
## 7   0.012 0.072 0.480 0.006 0.430
## 8   1.000 0.000 0.000 0.000 0.000
## 9   0.010 0.018 0.018 0.004 0.950
## 10  0.990 0.006 0.004 0.000 0.000
## 11  0.002 0.990 0.006 0.000 0.002
## 12  0.006 0.018 0.004 0.000 0.972
## 13  0.076 0.042 0.038 0.076 0.768
## 14  0.478 0.512 0.006 0.004 0.000
## 15  0.032 0.046 0.892 0.008 0.022
## 16  0.836 0.068 0.028 0.068 0.000
## 17  0.072 0.122 0.118 0.674 0.014
## 18  0.992 0.000 0.004 0.002 0.002
## 19  0.978 0.016 0.004 0.000 0.002
## 20  0.046 0.830 0.084 0.022 0.018
## 21  0.050 0.922 0.022 0.006 0.000
## 22  0.018 0.010 0.926 0.040 0.006
## 23  0.040 0.012 0.936 0.010 0.002
## 24  0.052 0.940 0.000 0.000 0.008
## 25  0.938 0.016 0.010 0.004 0.032
## 26  0.944 0.042 0.004 0.006 0.004
```

```
## 27  0.034 0.054 0.194 0.716 0.002
## 28  0.034 0.070 0.878 0.004 0.014
## 29  0.020 0.966 0.004 0.008 0.002
## 30  0.098 0.340 0.538 0.024 0.000
## 31  0.000 0.972 0.020 0.004 0.004
## 32  0.098 0.854 0.014 0.022 0.012
## 33  0.002 0.008 0.008 0.000 0.982
## 34  0.042 0.102 0.840 0.012 0.004
## 35  0.122 0.094 0.010 0.758 0.016
## 36  0.052 0.134 0.792 0.016 0.006
## 37  0.920 0.034 0.030 0.008 0.008
## 38  0.946 0.032 0.010 0.010 0.002
## 39  0.994 0.002 0.004 0.000 0.000
## 40  0.078 0.026 0.828 0.064 0.004
## 41  0.994 0.004 0.000 0.002 0.000
## 42  0.966 0.010 0.020 0.004 0.000
## 43  0.972 0.018 0.002 0.004 0.004
## 44  0.048 0.016 0.916 0.014 0.006
## 45  0.996 0.000 0.000 0.000 0.004
## 46  0.002 0.002 0.002 0.000 0.994
## 47  0.122 0.312 0.558 0.008 0.000
## 48  0.024 0.022 0.932 0.014 0.008
## 49  0.004 0.062 0.096 0.838 0.000
## 50  0.098 0.110 0.104 0.684 0.004
## 51  0.032 0.912 0.014 0.036 0.006
## 52  0.082 0.896 0.008 0.004 0.010
## 53  0.042 0.036 0.046 0.004 0.872
## 54  0.988 0.012 0.000 0.000 0.000
## 55  0.966 0.016 0.000 0.016 0.002
## 56  0.008 0.102 0.398 0.424 0.068
## 57  0.948 0.046 0.004 0.002 0.000
## 58  0.000 0.006 0.988 0.004 0.002
## 59  0.034 0.160 0.792 0.012 0.002
## 60  0.020 0.040 0.914 0.002 0.024
## 61  0.052 0.058 0.168 0.720 0.002
## 62  0.006 0.974 0.002 0.012 0.006
## 63  0.992 0.004 0.002 0.002 0.000
## 64  0.068 0.124 0.022 0.784 0.002
## 65  0.020 0.038 0.170 0.112 0.660
## 66  0.988 0.010 0.002 0.000 0.000
## 67  0.006 0.006 0.002 0.000 0.986
## 68  0.970 0.024 0.002 0.004 0.000
## 69  0.028 0.750 0.128 0.050 0.044
## 70  0.942 0.028 0.022 0.006 0.002
## 71  0.034 0.158 0.784 0.020 0.004
## 72  0.126 0.830 0.036 0.004 0.004
## 73  0.076 0.332 0.092 0.498 0.002
## 74  0.002 0.070 0.000 0.006 0.922
## 75  0.022 0.080 0.860 0.024 0.014
## 76  0.010 0.074 0.066 0.846 0.004
## 77  0.000 0.000 0.000 0.000 1.000
## 78  0.006 0.004 0.002 0.002 0.986
## 79  0.018 0.948 0.008 0.004 0.022
## 80  0.044 0.108 0.816 0.002 0.030
```

```
## 81  0.014 0.002 0.010 0.940 0.034
## 82  0.056 0.012 0.010 0.908 0.014
## 83  0.168 0.808 0.018 0.004 0.002
## 84  0.000 0.006 0.004 0.000 0.990
## 85  0.010 0.026 0.008 0.942 0.014
## 86  0.016 0.176 0.026 0.084 0.698
## 87  0.070 0.894 0.002 0.006 0.028
## 88  0.030 0.164 0.802 0.002 0.002
## 89  0.972 0.016 0.000 0.010 0.002
## 90  0.994 0.002 0.000 0.002 0.002
## 91  0.146 0.312 0.498 0.044 0.000
## 92  0.058 0.090 0.088 0.080 0.684
## 93  0.012 0.010 0.962 0.010 0.006
## 94  0.032 0.010 0.000 0.004 0.954
## 95  0.050 0.056 0.050 0.000 0.844
## 96  0.086 0.006 0.834 0.064 0.010
## 97  0.988 0.000 0.004 0.004 0.004
## 98  0.008 0.028 0.006 0.950 0.008
## 99  0.032 0.166 0.784 0.016 0.002
## 100 0.964 0.014 0.004 0.012 0.006
## 101 0.024 0.058 0.084 0.034 0.800
## 102 0.066 0.130 0.198 0.582 0.024
## 103 0.982 0.004 0.000 0.006 0.008
## 104 0.042 0.802 0.110 0.022 0.024
## 105 0.048 0.934 0.000 0.008 0.010
## 106 0.986 0.010 0.002 0.002 0.000
## 107 0.004 0.030 0.936 0.008 0.022
## 108 0.018 0.072 0.266 0.082 0.562
## 109 0.030 0.006 0.958 0.006 0.000
## 110 0.038 0.072 0.870 0.010 0.010
## 111 0.152 0.784 0.050 0.002 0.012
## 112 0.020 0.008 0.014 0.918 0.040
## 113 0.022 0.194 0.020 0.040 0.724
## 114 0.848 0.022 0.034 0.048 0.048
## 115 0.042 0.046 0.198 0.708 0.006
## 116 0.004 0.988 0.000 0.004 0.004
## 117 1.000 0.000 0.000 0.000 0.000
## 118 0.044 0.920 0.030 0.004 0.002
## 119 0.024 0.048 0.250 0.620 0.058
## 120 0.964 0.026 0.002 0.008 0.000
## 121 0.066 0.140 0.004 0.736 0.054
## 122 0.060 0.224 0.000 0.712 0.004
## 123 0.028 0.096 0.056 0.064 0.756
## 124 0.976 0.000 0.000 0.000 0.024
## 125 0.000 0.030 0.954 0.014 0.002
## 126 0.008 0.118 0.156 0.006 0.712
## 127 0.110 0.292 0.580 0.006 0.012
## 128 0.020 0.096 0.810 0.006 0.068
## 129 0.002 0.090 0.006 0.002 0.900
## 130 0.000 0.000 0.000 0.000 1.000
## 131 0.088 0.888 0.002 0.014 0.008
## 132 0.124 0.818 0.028 0.020 0.010
## 133 0.038 0.872 0.030 0.048 0.012
## 134 0.978 0.004 0.006 0.006 0.006
```

```
## 135 0.948 0.030 0.012 0.006 0.004
## 136 0.926 0.042 0.028 0.004 0.000
## 137 0.938 0.022 0.006 0.026 0.008
## 138 0.996 0.002 0.000 0.002 0.000
## 139 0.042 0.132 0.192 0.622 0.012
## 140 0.020 0.048 0.032 0.010 0.890
## 141 1.000 0.000 0.000 0.000 0.000
## 142 0.054 0.836 0.072 0.026 0.012
## 143 0.128 0.804 0.010 0.046 0.012
## 144 0.042 0.928 0.002 0.018 0.010
## 145 0.128 0.320 0.548 0.004 0.000
## 146 0.990 0.006 0.002 0.002 0.000
## 147 0.006 0.022 0.946 0.016 0.010
## 148 0.062 0.238 0.010 0.678 0.012
## 149 0.006 0.236 0.084 0.006 0.668
## 150 0.054 0.060 0.176 0.706 0.004
## 151 0.994 0.002 0.000 0.002 0.002
## 152 0.006 0.008 0.000 0.000 0.986
## 153 0.022 0.026 0.938 0.012 0.002
## 154 0.000 0.002 0.000 0.000 0.998
## 155 0.018 0.038 0.356 0.034 0.554
## 156 0.034 0.064 0.022 0.022 0.858
## 157 0.964 0.012 0.000 0.000 0.024
## 158 0.970 0.014 0.000 0.012 0.004
## 159 0.998 0.000 0.000 0.000 0.002
## 160 0.776 0.112 0.064 0.040 0.008
## 161 0.998 0.000 0.002 0.000 0.000
## 162 0.952 0.018 0.016 0.008 0.006
## 163 0.084 0.902 0.002 0.006 0.006
## 164 0.002 0.014 0.964 0.006 0.014
## 165 0.118 0.310 0.524 0.034 0.014
## 166 0.022 0.150 0.824 0.002 0.002
## 167 0.056 0.202 0.008 0.728 0.006
## 168 0.000 0.010 0.986 0.000 0.004
## 169 1.000 0.000 0.000 0.000 0.000
## 170 0.024 0.064 0.024 0.010 0.878
## 171 0.136 0.312 0.532 0.018 0.002
## 172 0.022 0.178 0.794 0.002 0.004
## 173 0.988 0.006 0.004 0.002 0.000
## 174 0.958 0.020 0.020 0.002 0.000
## 175 0.086 0.066 0.796 0.040 0.012
## 176 0.016 0.028 0.004 0.006 0.946
## 177 0.000 0.008 0.008 0.000 0.984
## 178 0.032 0.076 0.024 0.808 0.060
## 179 0.014 0.002 0.958 0.014 0.012
## 180 0.044 0.858 0.066 0.008 0.024
## attr(,"class")
## [1] "matrix" "array"  "votes"
```

‘ *The best model predicts A*