Machine Learning Project Notes going forward V5+

Chudnow

June 13, 2015

00. Checklist

[] R Markdown File on GITHUB [] < 2000 words [] < 5 Figures [] Address out of sample error and address via cross-validation [] HTML File on GITHUB [] Submit Prediction Files for Automatic Grading

https://github.com/AlanChudnow/DP/blob/gh-pages/ShinyProject_Slides.rmd.html
http://AlanChudnow.github.io/DP/ShinyProject_Slides.rmd.html

0. Background

Background

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement - a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Weightlifting Dataset from http://groupware.les.inf.puc-rio.br/har

This human activity recognition research has traditionally focused on discriminating between different activities, i.e. to predict "which" activity was performed at a specific point in time (like with the Daily Living Activities dataset above). The approach we propose for the Weight Lifting Exercises dataset is to investigate "how (well)" an activity was performed by the wearer. The "how (well)" investigation has only received little attention so far, even though it potentially provides useful information for a large variety of applications,such as sports training.

In this work (see the paper) we first define quality of execution and investigate three aspects that pertain to qualitative activity recognition: the problem of specifying correct execution, the automatic and robust detection of execution mistakes, and how to provide feedback on the quality of execution to the user. We tried out an on-body sensing approach (dataset here), but also an "ambient sensing approach" (by using Microsoft Kinect - dataset still unavailable)

Six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E).

Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes. Participants were supervised by an experienced weight lifter to make sure the execution complied to the manner they were supposed to simulate. The exercises were performed by six male participants aged between 20-28 years, with little weight lifting experience. We made sure that all participants could easily simulate the mistakes in a safe and controlled manner by using a relatively light dumbbell (1.25kg).

Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.

Read more: http://groupware.les.inf.puc-rio.br/har#ixzz3d4K34xch

Data

The training data for this project are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The test data are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har. If you use the document you create for this class for any purpose please cite them as they have been very generous in allowing their data to be used for this kind of assignment.

What you should submit

The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

1. Your submission should consist of a link to a Github repo with your ** R markdown** and **compiled HTML file** describing your analysis. Please constrain the text of the **writeup to < 2000 words** and the **number of figures to be less than 5.** It will make it easier for the graders if you submit a repo with a gh-pages branch so the HTML page can be viewed online (and you always want to make it easy on graders :-).

2. You should also *apply your machine learning algorithm to the 20 test cases available in the test data above*. Please *submit your predictions in appropriate format to the programming assignment for automated grading*. See the programming assignment for additional details.

Reproducibility

Due to security concerns with the exchange of R code, your code will not be run during the evaluation by your classmates. Please be sure that if they download the repo, they will be able to view the compiled HTML version of your analysis.

1. Read in the Data

```
library(caret)

## Loading required package: lattice
## Loading required package: ggplot2

SeedVersion <- 531

fn_Train <- "pml-training.csv"
fn_Test <- "pml-testing.csv"

df_rawTrain <-read.csv(fn_Train)  #Raw data directly from file
df_rawTest <- read.csv(fn_Test)   #Raw test data directly from file
```

2. Preliminary Exploration and Data Cleaning prior to Training

An examination of the data using R and Excel indicated that the training set had a number of factors that I didn't need to carry around for testing and training. Examples include:

- Col 1 Index (not relevant)
- Col 2 user_name (not relevant)
- Col 3:7 time and window number (not relevant)
- Col 12 kurosis (Almost all blank)
- Col 18 max_roll (almost all NA)

```
library(caret)

#Can I get rid of any columns first that don't change much
nzv <- nearZeroVar(df_rawTrain,saveMetrics=TRUE)
dropCol <- nzv$nzv

#Get rid of the first 7 columns are just indexes
dropCol[1:7] <- TRUE

#Identify and get rid of columns that are almost all NAs
countNA <- apply(df_rawTrain,2,function(x) {sum(is.na(x))})
dropNA <- countNA>(0.9*dim(df_rawTrain)[1])
```

```
dropCol[dropNA] <- TRUE

cleanPivot <- function(bigdf,dropCol){
    #This function will drop all the cols in my table
    #It will move the last col to be the first column
    sdf0 <- bigdf[,dropCol==FALSE]
    last <- length(sdf0)
    sdf1 <- data.frame(classe=sdf0[,last], sdf0[,1:(last-1)])
    return(sdf1)
}

df_train0 <- cleanPivot(df_rawTrain,dropCol) #Cleaned Training Set
df_TEST0 <- cleanPivot(df_rawTest, dropCol)  #Cleaned Final Test Set
```

3 Create Training/Test and Quiz Set for Cross-Validation

To design the prediction study, data is split into three sets

1. 60% Training - To provide data for machine learning (ML) algorithms
2. 20% Test set - To probe / test specific ML algorithms and settings
3. 20% Quiz set - To validate algorithms after selection

This worksheet is set up so that training and test sets can be randomly reassigned by changing a global `SeedVersion` value in the first block. This allows me to repeat the experiment with a different shuffle of rows into test and training sets, but leaving the Quiz set as is.

```
set.seed(135)
inTrain <- createDataPartition(df_train0[,1], p = 0.8)[[1]]

#Partion Data into BigTrain is for Training; quizing is for Validation
BigTrain   <- df_train0[ inTrain,];   quizing <- df_train0[-inTrain,]
BigTrain_u <- df_rawTrain[inTrain,2]; quiz_u  <- df_rawTrain[-inTrain,2]
BigTrain_c <- BigTrain[,1];           quiz_c  <- quizing[,1]

#Partion BigTrain into Training and Testing data as described above
set.seed(SeedVersion)
inTrain <- createDataPartition(BigTrain_c, p = 0.75)[[1]]

training <-BigTrain[ inTrain,];   testing <- BigTrain[-inTrain,]
train_u <- BigTrain_u[inTrain];   test_u <- BigTrain_u[-inTrain]
train_c <- training[,1];          test_c <- testing[,1]

colmax <- dim(training)[2]
colnames(training)

##  [1] "classe"               "roll_belt"            "pitch_belt"
##  [4] "yaw_belt"             "total_accel_belt"     "gyros_belt_x"
```

```
##  [7] "gyros_belt_y"         "gyros_belt_z"         "accel_belt_x"
## [10] "accel_belt_y"         "accel_belt_z"         "magnet_belt_x"
## [13] "magnet_belt_y"        "magnet_belt_z"        "roll_arm"
## [16] "pitch_arm"            "yaw_arm"              "total_accel_arm"
## [19] "gyros_arm_x"          "gyros_arm_y"          "gyros_arm_z"
## [22] "accel_arm_x"          "accel_arm_y"          "accel_arm_z"
## [25] "magnet_arm_x"         "magnet_arm_y"         "magnet_arm_z"
## [28] "roll_dumbbell"        "pitch_dumbbell"       "yaw_dumbbell"
## [31] "total_accel_dumbbell" "gyros_dumbbell_x"     "gyros_dumbbell_y"
## [34] "gyros_dumbbell_z"     "accel_dumbbell_x"     "accel_dumbbell_y"
## [37] "accel_dumbbell_z"     "magnet_dumbbell_x"    "magnet_dumbbell_y"
## [40] "magnet_dumbbell_z"    "roll_forearm"         "pitch_forearm"
## [43] "yaw_forearm"          "total_accel_forearm"  "gyros_forearm_x"
## [46] "gyros_forearm_y"      "gyros_forearm_z"      "accel_forearm_x"
## [49] "accel_forearm_y"      "accel_forearm_z"      "magnet_forearm_x"
## [52] "magnet_forearm_y"     "magnet_forearm_z"
```

4 Examine Training Data for any obvious features that can be exploited

4a. Plot the data for each dimension vs. column and color by class

```
library(manipulate)

myPlot <- function(cnum) {
    plot(1:length(training[,cnum]),training[,cnum],
         col=(as.numeric(training[,1])+1),
         ylab=colnames(training)[cnum],
         main=colnames(training)[cnum])
}

myBox <- function(cnum) {
    plot(training[,cnum] ~ training[,1],
         col=(as.numeric(training[,1])+1),
         ylab=colnames(training)[cnum],
         main=colnames(training)[cnum])
}
```

Note: These commands will allow for the user to graph each index. Just cut/paste into console window (after running hte R code above

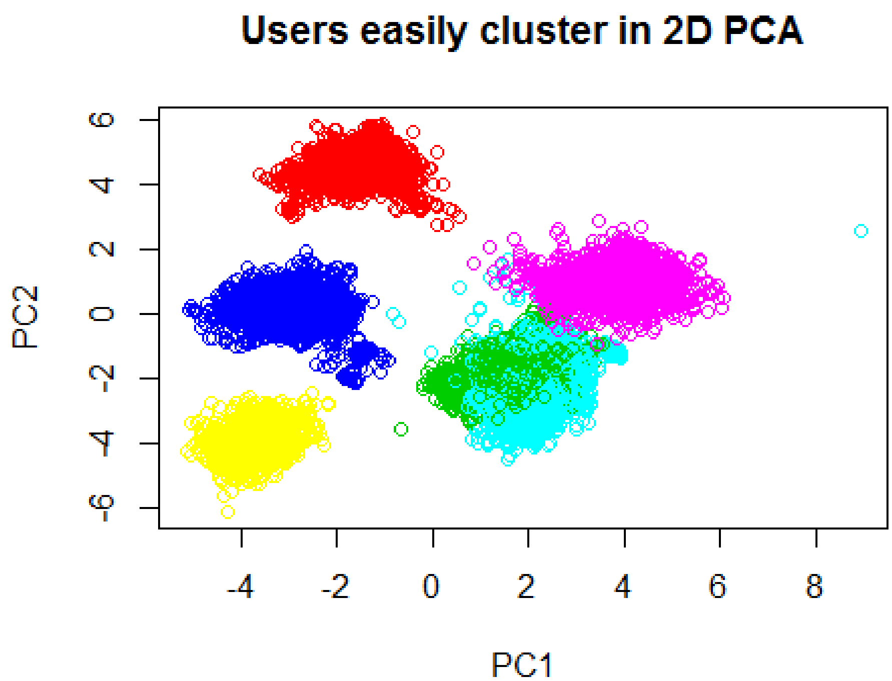
manipulate(myPlot(cnum),cnum=slider(2,dim(training)[2],step=1))
manipulate(myBox(cnum),cnum=slider(2,dim(training)[2],step=1))

Note: There are no obvious single columns that correlate well with states I have several rows that have clear outliers: gyros_dumbbell_x

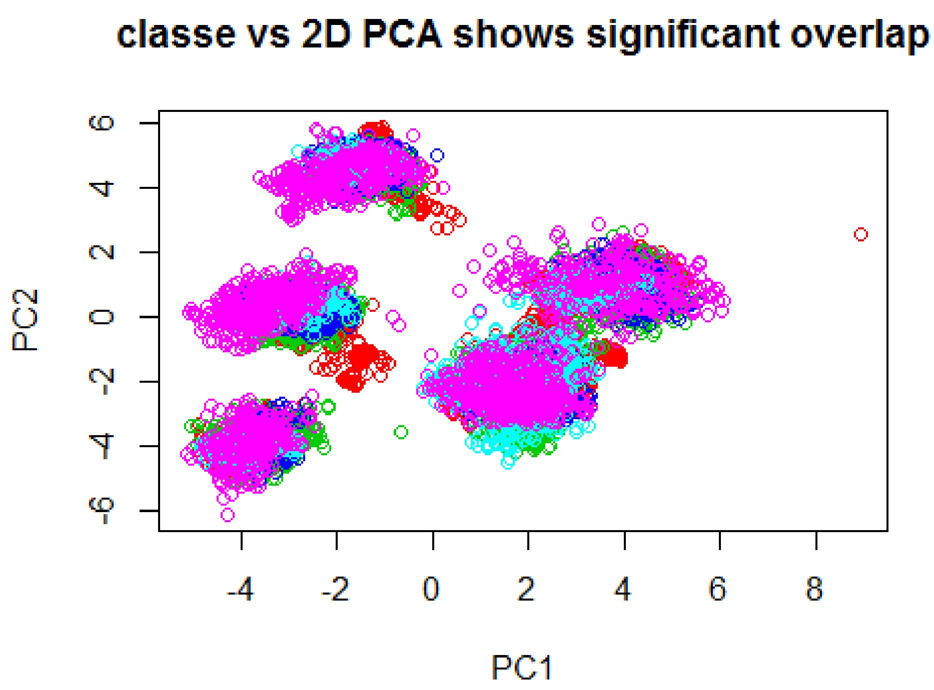
4b. Explore by seeing the Two dimensional SVD

It turns out that using an 2-D PCA analysis will form users into clusters quite readily but the relationship between different classes is not so obvious.

```
set.seed(SeedVersion)
preProc_pca2 <- preProcess(training[,-1], method="pca", pcaComp=2)
    #preProc_pca2
    #preProc_pca2$rotation

trainP_pca2 <- predict(preProc_pca2, training[,-1])
    plot(trainP_pca2$PC1,trainP_pca2$PC2, col=(as.numeric(train_u)+1),
         main="Users easily cluster in 2D PCA",
         xlab="PC1", ylab="PC2" )
```




```
    plot(trainP_pca2$PC1,trainP_pca2$PC2, col=(as.numeric(train_c)+1),
         main="classe vs 2D PCA shows significant overlap",
         xlab="PC1", ylab="PC2" )
```



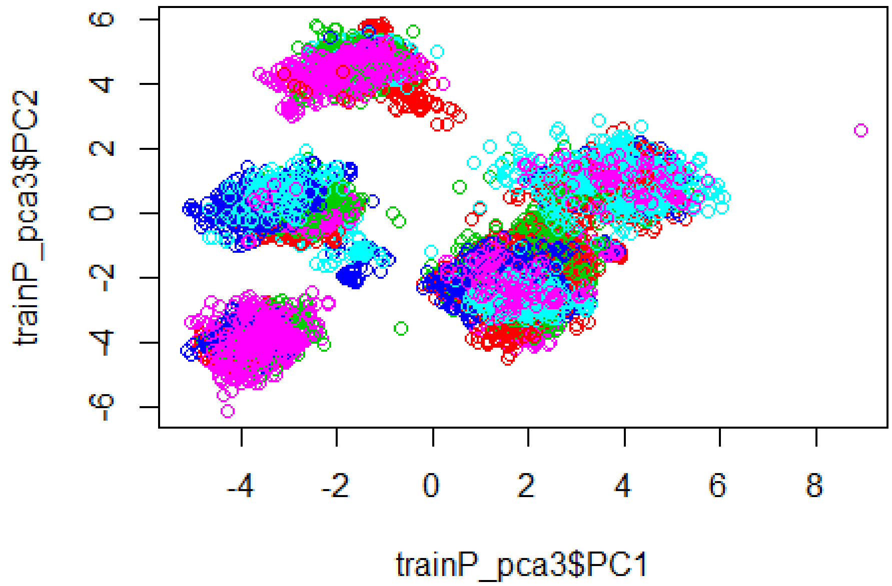
4c. Three dimensional PCA, just for fun

With a 3D PCA, there does appear to finally be a clustering of classe that we can take exploit (Although I did not code an algorithm to do this)

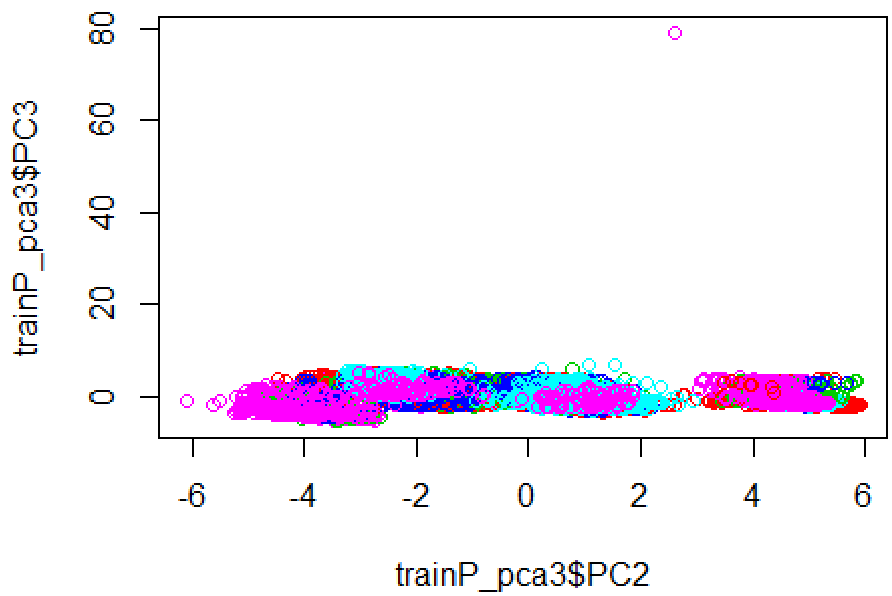
```
library(plot3D)

set.seed(SeedVersion)
preProc_pca3 <- preProcess(training[,-1], method="pca", pcaComp=3)
    preProc_pca3

##
## Call:
## preProcess.default(x = training[, -1], method = "pca", pcaComp = 3)
##
## Created from 11776 samples and 52 variables
## Pre-processing: principal component signal extraction, scaled, centered
##
## PCA used 3 components as specified.

    preProc_pca3$rotation

##                               PC1          PC2           PC3
## roll_belt            -0.314216660  0.112320507 -0.0696818603
## pitch_belt           -0.009997571 -0.293653194 -0.0620158342
## yaw_belt             -0.214121793  0.241978670 -0.0227350802
## total_accel_belt     -0.310031711  0.092493347 -0.0889471506
```

```
## gyros_belt_x          0.082880883  0.200825387  0.1823808972
## gyros_belt_y         -0.112904600  0.200673249  0.0813952211
## gyros_belt_z          0.177409772  0.055642449  0.1007880354
## accel_belt_x         -0.004968400  0.295425922  0.0771485703
## accel_belt_y         -0.318559136  0.018912048 -0.0929064792
## accel_belt_z          0.321755960 -0.087433472  0.0660643553
## magnet_belt_x        -0.030437066  0.288972219  0.0332690070
## magnet_belt_y         0.110008462  0.096307617 -0.0629749266
## magnet_belt_z         0.049930521  0.126815461 -0.0508887444
## roll_arm              0.074099465 -0.174707428  0.0535941281
## pitch_arm             0.031295254  0.062900834 -0.2282547068
## yaw_arm               0.059075194 -0.116130353  0.0049416601
## total_accel_arm       0.115330414 -0.028212742  0.0672095410
## gyros_arm_x          -0.018815703  0.050949729  0.0070629391
## gyros_arm_y           0.084238713 -0.076320989 -0.0142399432
## gyros_arm_z          -0.170806889  0.173850964  0.0713833944
## accel_arm_x          -0.152648580 -0.112050961  0.1729005351
## accel_arm_y           0.274602092 -0.111526032 -0.1332470782
## accel_arm_z          -0.126262781 -0.012632724 -0.2703727581
## magnet_arm_x         -0.085986445 -0.007804172  0.2605648419
## magnet_arm_y          0.061160356  0.024485202 -0.3602038376
## magnet_arm_z          0.028933921  0.023790275 -0.3031478769
## roll_dumbbell         0.079936613  0.134757627  0.0515313696
## pitch_dumbbell       -0.102096355 -0.153924953  0.0871603833
## yaw_dumbbell         -0.110941815 -0.273868236  0.0103108799
## total_accel_dumbbell  0.161853944  0.156677246 -0.1148010556
## gyros_dumbbell_x     -0.007588686 -0.008409737 -0.1761646940
## gyros_dumbbell_y      0.001775724  0.036972163  0.1353274842
## gyros_dumbbell_z      0.005235337  0.004038968  0.1623840551
## accel_dumbbell_x     -0.162923890 -0.147325906  0.1240453461
## accel_dumbbell_y      0.171568901  0.192350123 -0.0039761648
## accel_dumbbell_z     -0.139125914 -0.256149796  0.0712463350
## magnet_dumbbell_x    -0.156665455 -0.205756913 -0.1332432309
## magnet_dumbbell_y     0.134448681  0.180423179  0.1951721177
## magnet_dumbbell_z     0.174427259 -0.014681074  0.1674793947
## roll_forearm          0.065119348 -0.042269449 -0.1425815564
## pitch_forearm        -0.138273741 -0.110639536  0.0978974682
## yaw_forearm           0.113866269 -0.031757826 -0.1256927063
## total_accel_forearm  -0.004318224  0.098028179 -0.0002941761
## gyros_forearm_x      -0.081814777  0.184133903 -0.1316302762
## gyros_forearm_y       0.001721299  0.018839237  0.1529849508
## gyros_forearm_z       0.006874491  0.020445575  0.1659692995
## accel_forearm_x       0.195991660 -0.080225012 -0.1199722572
## accel_forearm_y       0.029587841  0.092678027 -0.1096069350
## accel_forearm_z      -0.032203873  0.034216439  0.1956415862
## magnet_forearm_x      0.105789742 -0.005705145  0.0069349130
## magnet_forearm_y      0.020408446  0.050580376 -0.1300121569
## magnet_forearm_z     -0.047318897  0.109683156 -0.1783523220
```

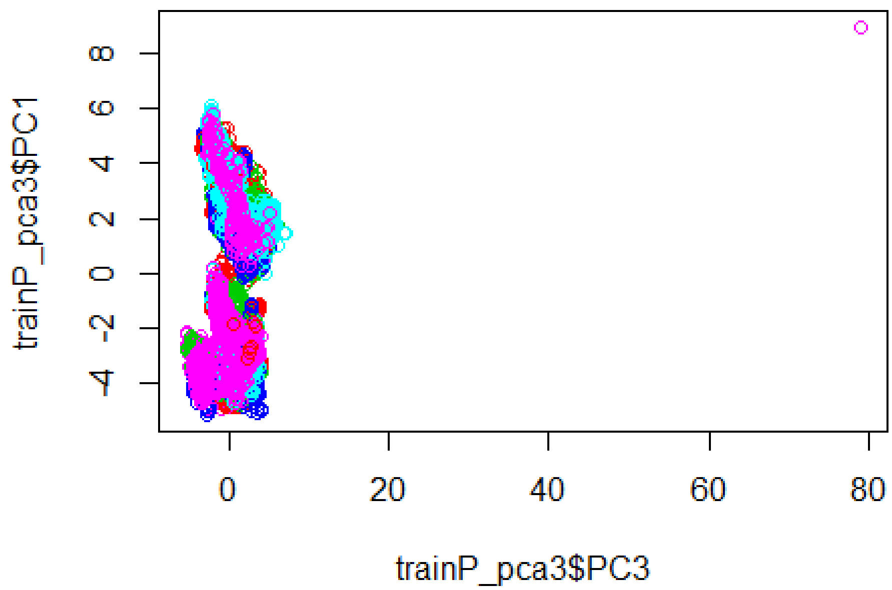
```
trainP_pca3 <- predict(preProc_pca3, training[,-1])
    plot(trainP_pca3$PC1,trainP_pca3$PC2, col=(as.numeric(test_c)+1))
```



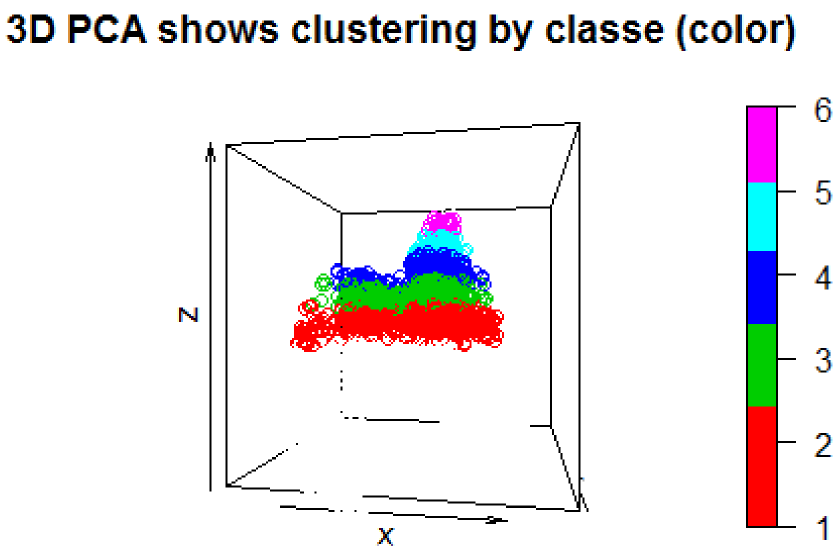

```
    plot(trainP_pca3$PC2,trainP_pca3$PC3, col=(as.numeric(test_c)+1))
```



```r
plot(trainP_pca3$PC3,trainP_pca3$PC1, col=(as.numeric(test_c)+1))
```




```r
scatter3D(trainP_pca3$PC1,trainP_pca3$PC2, trainP_pca3$PC3,
          xlim=c(-8,8), ylim=c(-5,7),zlim=c(-4,8),clim=c(1,6),
          main="3D PCA shows clustering by classe (color)",
       col=(as.numeric(test_c)+1),phi=0,theta=10)
```




```
my3D<- function(v_phi,v_theta) {
    scatter3D(trainP_pca3$PC1,
              trainP_pca3$PC2,
              trainP_pca3$PC3,
              xlim=c(-8,8), ylim=c(-6,7),zlim=c(-4,8),clim=c(1,6),
              xlab="PC1",ylab="PC2",zlab="PC3",
              phi=v_phi,
              theta=v_theta,
              col=(as.numeric(train_c)+1))
}
```

manipulate(my3D(v_phi,v_theta), v_phi=slider(-90,90,step=5,initial=0), v_theta=slider(-90,90,step=5,initial=10))

5. Using Machine learning models to predict classe

5A. KNN Techniques (Zero Mean/ 1 sigma) *Accuracy : 0.98*

kNN has many useful characteristics, one of which being its insensitivity to outliers that makes it resilient to any errors in the classification data (the supervised learning phase). As a downside, the algorithm is noted for its CPU and memory greediness. For this example we scale all the columns for mean 0 and sd=1 prior to running the algorithm.

```
Confusion Matrix and Statistics

          Reference
```

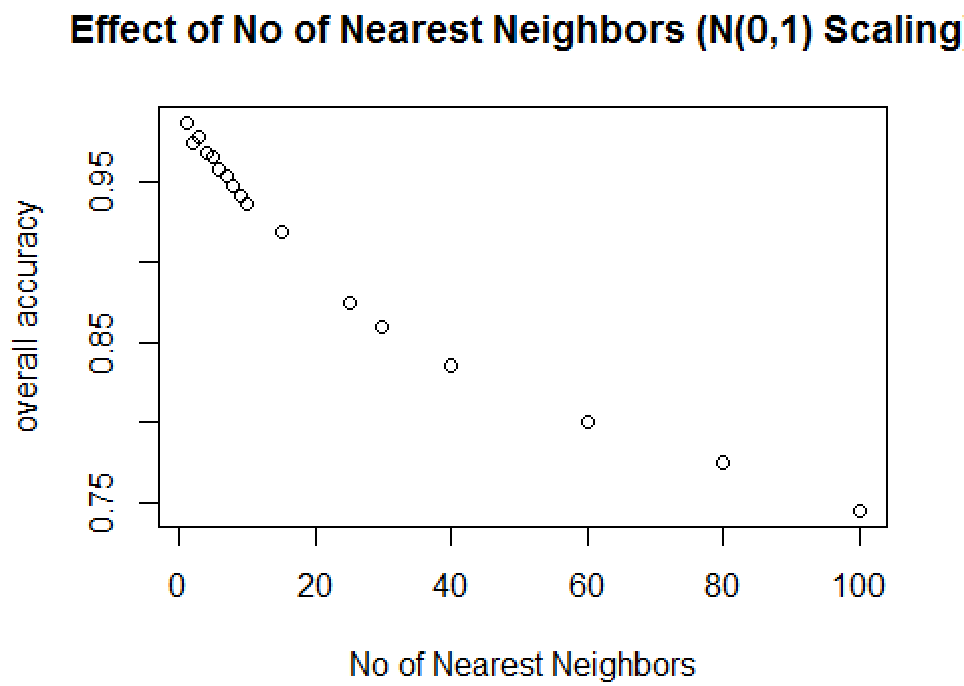
```
Prediction    A    B    C    D    E
         A 1105    4    5    2    0
         B   14  729   14    1    1
         C    3    7  665    8    1
         D    0    1   12  627    3
         E    1    3    4    4  709

Overall Statistics:

               Accuracy : 0.9776
                 95% CI : (0.9724, 0.982)
```

```
library (class)

znorm <- function(x) { return ((x - mean(x)) / sd(x))}

x<- znorm(rnorm(10,mean=30,sd=5)); mean(x); sd(x)  #Check Code
```

```
## [1] 6.071532e-17
```

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

```
train_knnZ <- as.data.frame(lapply(training[2:colmax],znorm))
test_knnZ <-  as.data.frame(lapply(testing[2:colmax],znorm))

#Loop over different values of K to see which works best

x <- c(1,2,3,4,5,6,7,8,9,10,15,25,30,40,60,80,100) #valued of K
y <- x * 0

set.seed(SeedVersion)
for(nNN in 1:length(x)){
    knn_test_pred <- knn(train = train_knnZ, #Training Set
                     test = test_knnZ,  #Test Data
                      cl = train_c, #Truth Labels for Training Data
                      k=x[nNN]) #Number of valued to compare
    cl_knn<- confusionMatrix(test_c,knn_test_pred)
    y[nNN] <- cl_knn$overall[1]
}

plot(x,y,xlab="No of Nearest Neighbors",
     ylab="overall accuracy",
     main="Effect of No of Nearest Neighbors (N(0,1) Scaling)")
```




```
y

##  [1] 0.9869997 0.9737446 0.9785878 0.9676268 0.9648228 0.9579404 0.9536069
##  [8] 0.9477441 0.9408616 0.9370380 0.9184298 0.8743309 0.8590365 0.8360948
## [15] 0.8001529 0.7749172 0.7448381

kBest = x[which(y==max(y))]

#set kBest<-3 Anyway because 1 seems overtraining
kBest <-3

knn_test_pred <- knn(train = train_knnZ, test = test_knnZ,
                     cl = train_c, k=kBest)

cl_knn<- confusionMatrix(test_c,knn_test_pred)
cl_knn

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1100   11    4    0    1
##          B   11  735   12    1    0
##          C    3    8  669    4    0
##          D    0    0   15  627    1
##          E    0    6    3    5  707
##
```

```
## Overall Statistics
##
##                Accuracy : 0.9783
##                  95% CI : (0.9733, 0.9827)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9726
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9874   0.9671   0.9516   0.9843   0.9972
## Specificity            0.9943   0.9924   0.9953   0.9951   0.9956
## Pos Pred Value         0.9857   0.9684   0.9781   0.9751   0.9806
## Neg Pred Value         0.9950   0.9921   0.9895   0.9970   0.9994
## Prevalence             0.2840   0.1937   0.1792   0.1624   0.1807
## Detection Rate         0.2804   0.1874   0.1705   0.1598   0.1802
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9909   0.9798   0.9735   0.9897   0.9964
```

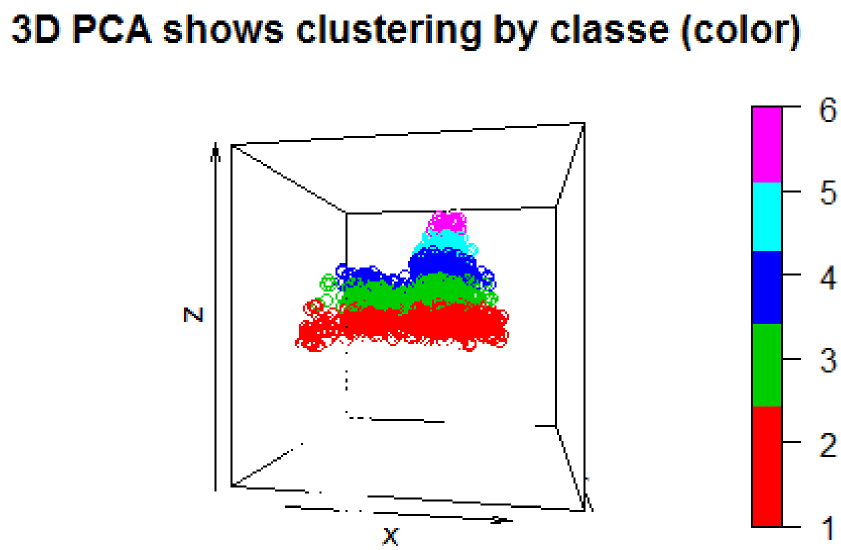
```
cl_knn$overall[1]
```

```
##  Accuracy
## 0.9783329
```

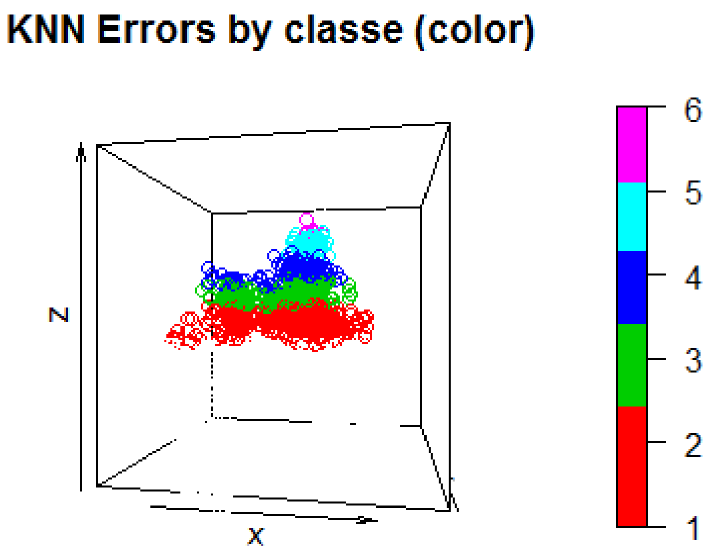
```
kBest
```

```
## [1] 3
```

```
erroring <- testing[-(test_c==knn_test_pred),]
error_pca3 <- predict(preProc_pca3, erroring[,-1])

    scatter3D(trainP_pca3$PC1,trainP_pca3$PC2, trainP_pca3$PC3,
              xlim=c(-8,8), ylim=c(-5,7),zlim=c(-4,8),clim=c(1,6),
              main="3D PCA shows clustering by classe (color)",
           col=(as.numeric(test_c)+1),phi=0,theta=10)
```




```
scatter3D(error_pca3$PC1,error_pca3$PC2, error_pca3$PC3,
          xlim=c(-8,8), ylim=c(-5,7),zlim=c(-4,8),clim=c(1,6),
          main="KNN Errors by classe (color)",
       col=(as.numeric(erroring[,1])+1),phi=0,theta=10)
```




```
flip3D<- function(v_phi,v_theta,flip) {
    if(flip==0){
            scatter3D(trainP_pca3$PC1,
              trainP_pca3$PC2,
              trainP_pca3$PC3,
              xlim=c(-8,8), ylim=c(-6,7),zlim=c(-4,8),clim=c(1,6),
              main="DATA", xlab="PC1",ylab="PC2",zlab="PC3",
              phi=v_phi,
              theta=v_theta,
              col=(as.numeric(train_c)+1))
    }
    if(flip==1){
            scatter3D(error_pca3$PC1,
              error_pca3$PC2,
              error_pca3$PC3,
              xlim=c(-8,8), ylim=c(-6,7),zlim=c(-4,8),clim=c(1,6),
              main="ERROR", xlab="PC1",ylab="PC2",zlab="PC3",
              phi=v_phi,
              theta=v_theta,
              col=(as.numeric(erroring[,1])+1))

    }

}

#manipulate(flip3D(v_phi,v_theta,flip),
```

```
#           v_phi=slider(-90,90,step=5,initial=0),
#           v_theta=slider(-90,90,step=5,initial=10),
#           flip=slider(0,1,step=1,initial=0)
#           )

#try it on the QUIZ SET - But Don't Look Yet
Quiz_knnZ <- as.data.frame(lapply(quizing[2:colmax],znorm))
knnN0_Quiz_pred <- knn(train = train_knnZ, #Training Set
                     test = Quiz_knnZ,  #Test Data
                     cl = train_c, #Truth Labels for Training Data
                     k=kBest) #Number of valued to compare
knn_cl_quiz <- confusionMatrix(quiz_c, knnN0_Quiz_pred)

#Try it on the FINAL TEST SET - And Save Answers for later output
TEST0_knnZ <- as.data.frame(lapply(df_TEST0[2:colmax],znorm))
knnN0_TEST0_pred <- knn(train = train_knnZ, #Training Set
                     test = TEST0_knnZ,  #Test Data
                     cl = train_c, #Truth Labels for Training Data
                     k=kBest) #Number of valued to compare
```

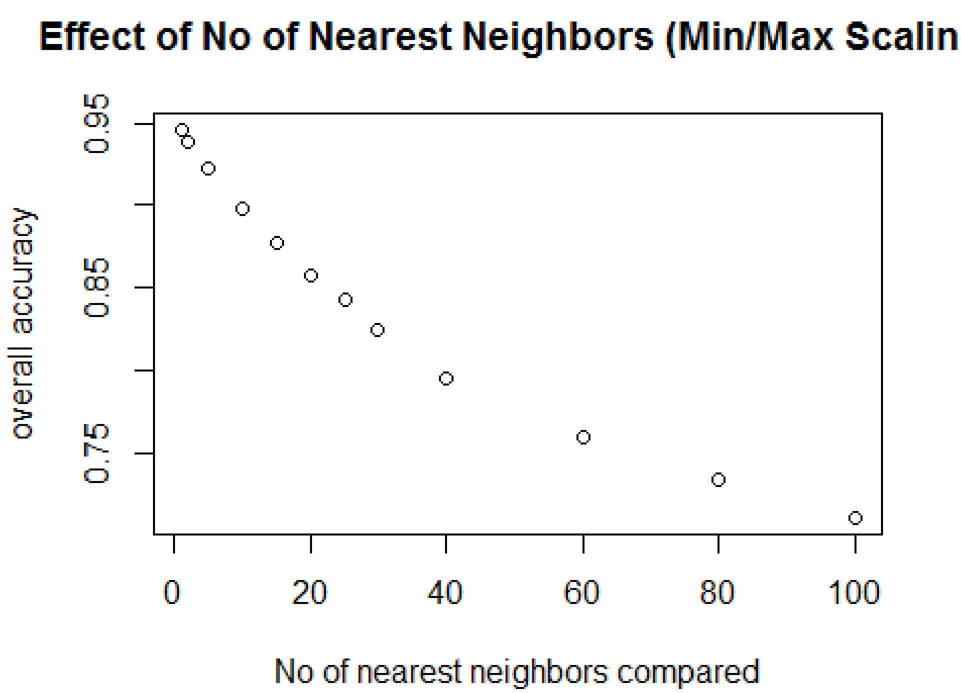
5b. KNN Techniques (Min/Max Scaling) *Accuracy : 0.94%*

For this example we scale all the columns for mean 0 and sd=1 prior to running the algorithm. Because I have outliers, this may do worse than the approach N(0,1) above because the outliers will squeeze the data.

```
Reference
Prediction    A    B    C    D    E
         A 1091   18    3    4    0
         B   22  707   24    1    5
         C    3   26  608   37   10
         D    1    3   25  610    4
         E    0    4   11   10  696

Overall Statistics

               Accuracy : 0.9462
                 95% CI : (0.9387, 0.9531)
```

```
library (class)

normalize <- function(x) { return ((x - min(x)) / (max(x) - min(x)))}

normalize(c(1, 2, 3, 4, 5))  #Check Code

## [1] 0.00 0.25 0.50 0.75 1.00

normalize(c(10, 20, 30, 40, 50)) #Check Code

## [1] 0.00 0.25 0.50 0.75 1.00
```

```
#Create a set of scaled test/training sets for the algorithm#
#In this case all data is between scaled to between 0 and 1

train_knnMM <- as.data.frame(lapply(training[2:colmax],normalize))
test_knnMM <-  as.data.frame(lapply(testing[2:colmax],normalize))
#remember train_c <- training[,1]; #Labels for training
#remember test_c <- testing[,1] #Labels for testing

#Try this at a number of different nearest neighbors and pick the best
x <- c(1,2,5,10,15,20,25,30,40,60,80,100)
y <- x * 0

set.seed(SeedVersion)
for(nNN in 1:length(x)){
    knn_test_pred <- knn(train = train_knnMM, #Training Set
                     test = test_knnMM,  #Test Data
                      cl = train_c, #Truth Labels for Training Data
                      k=x[nNN]) #Number of valued to compare

    cl_knn<- confusionMatrix(test_c,knn_test_pred)
    y[nNN] <- cl_knn$overall[1]
}

plot(x,y,xlab="No of nearest neighbors compared",
     ylab="overall accuracy",
     main="Effect of No of Nearest Neighbors (Min/Max Scaling)")
```



```
y

##  [1] 0.9459597 0.9383125 0.9230181 0.8980372 0.8766250 0.8580168 0.8429773
##  [8] 0.8243691 0.7947999 0.7596227 0.7341320 0.7106806

kBest = x[which(y==max(y))]
knn_test_pred <- knn(train = train_knnMM, #Training Set
                     test = test_knnMM,  #Test Data
                      cl = train_c, #Truth Labels for Training Data
                      k=kBest) #Number of valued to compare



#try it on the QUIZ SET But Don't Look Yet
Quiz_knnMM <- as.data.frame(lapply(quizing[2:colmax],znorm))
knnMM_Quiz_pred <- knn(train = train_knnMM, #Training Set
                     test = Quiz_knnMM,  #Test Data
                     cl = train_c, #Truth Labels for Training Data
                     k=kBest) #Number of valued to compare

#try it on the TEST SET But Don't Look Yet.
TEST0_knnMM <- as.data.frame(lapply(df_TEST0[2:colmax],normalize))
knnMM_TEST0_pred <- knn(train = train_knnMM, #Training Set
                     test = TEST0_knnMM,  #Test Data
                     cl = train_c, #Truth Labels for Training Data
                     k=kBest) #Number of valued to compare

cl_knn<- confusionMatrix(test_c,knn_test_pred)
cl_knn

## Confusion Matrix and Statistics
##
##             Reference
## Prediction    A    B    C    D    E
##          A 1093   11    4    6    2
##          B   29  701   25    1    3
##          C    0   28  613   30   13
##          D    0    2   28  610    3
##          E    0    4    8   15  694
##
## Overall Statistics
##
##                Accuracy : 0.946
##                  95% CI : (0.9384, 0.9528)
##     No Information Rate : 0.286
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9316
##  Mcnemar's Test P-Value : 0.0008549
##
## Statistics by Class:
```

```
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9742   0.9397   0.9041   0.9215   0.9706
## Specificity            0.9918   0.9817   0.9781   0.9899   0.9916
## Pos Pred Value         0.9794   0.9236   0.8962   0.9487   0.9626
## Neg Pred Value         0.9897   0.9858   0.9799   0.9841   0.9934
## Prevalence             0.2860   0.1902   0.1728   0.1687   0.1823
## Detection Rate         0.2786   0.1787   0.1563   0.1555   0.1769
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9830   0.9607   0.9411   0.9557   0.9811
```

```
cl_knn$overall[1]
```

```
##  Accuracy
## 0.9459597
```

```
kBest
```

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

5c. NaiveBayes *Accuracy : 49%*

This algorithms computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule. Bayes performed especially poorly on this example, although better than random guessing.

```
Confusion Matrix and Statistics

          Reference
Prediction   A   B   C   D   E
         A 319  93 494 179  31
         B  19 456 175  54  55
         C   7  54 484 104  35
         D   0  18 235 317  73
         E  12 150 113  98 348

Overall Statistics

               Accuracy : 0.4904
                 95% CI : (0.4747, 0.5062)
    No Information Rate : 0.3826
```

```
require(e1071)
```

```
## Loading required package: e1071
```

```
set.seed(SeedVersion)
m_fitBayes <- naiveBayes(classe ~ ., data=training, laplace = 0)
bayes_test_pred <- predict(m_fitBayes,testing,type="class" )
cl_bayes<- confusionMatrix(test_c,bayes_test_pred)
cl_bayes$overall[1]
```

```
##  Accuracy
## 0.4980882

bayes_Quiz_pred <- predict(m_fitBayes,quizing,type="class")
bayes_TEST0_pred <- predict(m_fitBayes,df_TEST0,type="class" )
```

5d. Logistical Regression *Accuracy 74%*

Logistic Regression is a classification method that models the probability of an observation belonging to one of two classes. As such, normally logistic regression is demonstrated with binary classification problem (2 classes). Logistic Regression can also be used on problems with more than two classes (multinomial), as in this case.

See http://machinelearningmastery.com/linear-classification-in-r/

```
Confusion Matrix and Statistics

          Reference
Prediction   A   B   C   D   E
         A 965  33  62  47   9
         B  97 501  79  15  67
         C  66  62 479  45  32
         D  32  27  68 486  30
         E  31 105  42  68 475

Overall Statistics

               Accuracy : 0.7408
                 95% CI : (0.7267, 0.7544)

library(VGAM)

## Loading required package: stats4
## Loading required package: splines
##
## Attaching package: 'VGAM'
##
## The following object is masked from 'package:caret':
##
##     predictors

set.seed(SeedVersion)
fit_vgam <- vglm(classe~., family=multinomial, data=training)

## Warning in checkwz(wz, M = M, trace = trace, wzepsilon = control
## $wzepsilon): 26 elements replaced by 1.819e-12

## Warning in checkwz(wz, M = M, trace = trace, wzepsilon = control
## $wzepsilon): 45 elements replaced by 1.819e-12

## Warning in checkwz(wz, M = M, trace = trace, wzepsilon = control
## $wzepsilon): 74 elements replaced by 1.819e-12
```

```
## Warning in checkwz(wz, M = M, trace = trace, wzepsilon = control
## $wzepsilon): 79 elements replaced by 1.819e-12

## Warning in checkwz(wz, M = M, trace = trace, wzepsilon = control
## $wzepsilon): 79 elements replaced by 1.819e-12

## Warning in checkwz(wz, M = M, trace = trace, wzepsilon = control
## $wzepsilon): 79 elements replaced by 1.819e-12
```

```
#summary(fit_vgam)

prob_vgam <- predict(fit_vgam, testing, type="response")
pred_vgam <- apply(prob_vgam, 1, which.max)
pred_vgam[which(pred_vgam=="1")] <- levels(testing$classe)[1]
pred_vgam[which(pred_vgam=="2")] <- levels(testing$classe)[2]
pred_vgam[which(pred_vgam=="3")] <- levels(testing$classe)[3]
pred_vgam[which(pred_vgam=="4")] <- levels(testing$classe)[4]
pred_vgam[which(pred_vgam=="5")] <- levels(testing$classe)[5]

cl_vgam<- confusionMatrix(test_c,pred_vgam)
cl_vgam
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   A   B   C   D   E
##          A 962  35  58  52   9
##          B  93 490  81  24  71
##          C  54  37 520  44  29
##          D  41  22  63 484  33
##          E  28 105  48  64 476
## 
## Overall Statistics
## 
##                Accuracy : 0.7474
##                  95% CI : (0.7335, 0.7609)
##     No Information Rate : 0.3003
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.68
##  Mcnemar's Test P-Value : 9.815e-13
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8166   0.7112   0.6753   0.7246   0.7702
## Specificity            0.9439   0.9168   0.9480   0.9512   0.9259
## Pos Pred Value         0.8620   0.6456   0.7602   0.7527   0.6602
## Neg Pred Value         0.9230   0.9371   0.9228   0.9439   0.9557
## Prevalence             0.3003   0.1756   0.1963   0.1703   0.1575
## Detection Rate         0.2452   0.1249   0.1326   0.1234   0.1213
```

```
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.8803   0.8140   0.8117   0.8379   0.8480
```

```
cl_vgam$overall[1]
```

```
##  Accuracy 
## 0.7473872
```

```
pq_vgam <- predict(fit_vgam, quizing ,type="response")
vgam_quiz_pred <- apply(pq_vgam, 1, which.max)
vgam_quiz_pred[which(vgam_quiz_pred=="1")] <- levels(testing$classe)[1]
vgam_quiz_pred[which(vgam_quiz_pred=="2")] <- levels(testing$classe)[2]
vgam_quiz_pred[which(vgam_quiz_pred=="3")] <- levels(testing$classe)[3]
vgam_quiz_pred[which(vgam_quiz_pred=="4")] <- levels(testing$classe)[4]
vgam_quiz_pred[which(vgam_quiz_pred=="5")] <- levels(testing$classe)[5]

pTEST0_vgam <- predict(fit_vgam, df_TEST0 ,type="response")
vgam_TEST0_pred <- apply(pTEST0_vgam, 1, which.max)
vgam_TEST0_pred[which(vgam_TEST0_pred=="1")] <- levels(testing$classe)[1]
vgam_TEST0_pred[which(vgam_TEST0_pred=="2")] <- levels(testing$classe)[2]
vgam_TEST0_pred[which(vgam_TEST0_pred=="3")] <- levels(testing$classe)[3]
vgam_TEST0_pred[which(vgam_TEST0_pred=="4")] <- levels(testing$classe)[4]
vgam_TEST0_pred[which(vgam_TEST0_pred=="5")] <- levels(testing$classe)[5]
```

5e. Linear Discriminant Analysis *Accuracy 69%*

LDA is a classification method that finds a linear combination of data attributes that best separate the data into classes.

See http://machinelearningmastery.com/linear-classification-in-r/

```
Confusion Matrix and Statistics

          Reference
Prediction   A   B   C   D   E
         A 906  31  92  85   2
         B 126 467  99  24  43
         C  85  63 439  80  17
         D  28  25  66 497  27
         E  24 131  54  84 428

Overall Statistics
                                          
               Accuracy : 0.6977          
                 95% CI : (0.683, 0.712)
```

```
library(MASS)

set.seed(SeedVersion)
fit_lda <- lda(classe~., data=training)
summary(fit_lda)
```

```
##         Length Class  Mode
## prior     5    -none- numeric
## counts    5    -none- numeric
## means   260    -none- numeric
## scaling 208    -none- numeric
## lev       5    -none- character
## svd       4    -none- numeric
## N         1    -none- numeric
## call      3    -none- call
## terms     3    terms  call
## xlevels   0    -none- list
```

```
pred_lda <- predict(fit_lda, testing)$class
cl_lda<- confusionMatrix(test_c,pred_lda)
cl_lda
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 920  27  84  82   3
##          B 113 477 102  33  34
##          C  49  49 475  95  16
##          D  42  15  68 499  19
##          E  20 126  70  76 429
##
## Overall Statistics
##
##                Accuracy : 0.7137
##                  95% CI : (0.6993, 0.7278)
##     No Information Rate : 0.2916
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6381
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8042   0.6873   0.5945   0.6357   0.8563
## Specificity            0.9295   0.9127   0.9331   0.9541   0.9147
## Pos Pred Value         0.8244   0.6285   0.6944   0.7760   0.5950
## Neg Pred Value         0.9202   0.9314   0.9000   0.9128   0.9775
## Prevalence             0.2916   0.1769   0.2037   0.2001   0.1277
## Detection Rate         0.2345   0.1216   0.1211   0.1272   0.1094
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.8668   0.8000   0.7638   0.7949   0.8855
```

```
cl_lda$overall[1]
```

```
##  Accuracy
## 0.7137395

lda_quiz_pred <- predict(fit_lda,quizing)$class
lda_TEST0_pred <- predict(fit_lda, df_TEST0)$class
```

5f. Partial Least Squares Discriminant Analysis *Accuracy 39%%*

Partial Least Squares Discriminate Analysis is the application of LDA on a dimension-reducing projection of the input data (partial least squares).

See http://machinelearningmastery.com/linear-classification-in-r/

Confusion Matrix and Statistics

```
Reference
Prediction   A   B   C   D   E
         A 587  89 233 176  31
         B 108 290 172 135  54
         C 153 118 287 104  22
         D  33  81 172 307  50
         E  96 185 180 169  91

Overall Statistics

               Accuracy : 0.3982
                 95% CI : (0.3828, 0.4137)
```

```
library(caret)

train_plsda <- training[,2:colmax]
test_plsda <- testing[,2:colmax]

set.seed(SeedVersion)
fit_plsda <- plsda(train_plsda,train_c, probMethod="Bayes")
pred_plsda <- predict(fit_plsda, test_plsda)
cl_plsda<- confusionMatrix(test_c,pred_plsda)
cl_plsda

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 630  94 208 152  32
##          B 111 279 186 124  59
##          C 145 109 290 115  25
##          D  31  93 177 290  52
##          E 105 179 167 161 109
##
## Overall Statistics
##
```

```
##                Accuracy : 0.4073
##                  95% CI : (0.3919, 0.4229)
##     No Information Rate : 0.262
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2544
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.6164  0.37003  0.28210  0.34442  0.39350
## Specificity            0.8325  0.84853  0.86390  0.88543  0.83214
## Pos Pred Value         0.5645  0.36759  0.42398  0.45101  0.15118
## Neg Pred Value         0.8603  0.84987  0.77215  0.83171  0.94753
## Prevalence             0.2605  0.19220  0.26204  0.21463  0.07061
## Detection Rate         0.1606  0.07112  0.07392  0.07392  0.02778
## Detection Prevalence   0.2845  0.19347  0.17436  0.16391  0.18379
## Balanced Accuracy      0.7245  0.60928  0.57300  0.61492  0.61282
```

```
cl_plsda$overall[1]
```

```
##  Accuracy
## 0.4073413
```

6. Validate the best algorithm on quiz data

The KNN Algorithm using N0,1) scaling proved to be the best algorithm. Here it is running on the Quiz data:

``` Confusion Matrix and Statistics

```
      Reference
```

Prediction A B C D E A 1108 7 1 0 0 B 10 734 12 0 3 C 0 5 670 9 0 D 0 0 21 622 0 E 3 4 4 2 708

Overall Statistics

```
           Accuracy : 0.9794
             95% CI : (0.9744, 0.9836)
```

```
knn_cl_quiz <- confusionMatrix(quiz_c, knnN0_Quiz_pred)
knn_cl_quiz
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1105   10    1    0    0
##          B   10  738   10    1    0
##          C    0    5  666   13    0
```
```

```
##          D    0    0   23  618    2
##          E    4    4    3    3  707
##
## Overall Statistics
##
##                Accuracy : 0.9773
##                  95% CI : (0.9722, 0.9817)
##     No Information Rate : 0.2852
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9713
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9875   0.9749   0.9474   0.9732   0.9972
## Specificity            0.9961   0.9934   0.9944   0.9924   0.9956
## Pos Pred Value         0.9901   0.9723   0.9737   0.9611   0.9806
## Neg Pred Value         0.9950   0.9940   0.9886   0.9948   0.9994
## Prevalence             0.2852   0.1930   0.1792   0.1619   0.1807
## Detection Rate         0.2817   0.1881   0.1698   0.1575   0.1802
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9918   0.9841   0.9709   0.9828   0.9964
```

6. Stack the Test Results from each model Just for fun and to compare

```
stackResults = data.frame(
    knnN0 = knnN0_TEST0_pred,
    knnMM = knnMM_TEST0_pred,
    bayes = bayes_TEST0_pred,
    vgam = vgam_TEST0_pred,
    lda = lda_TEST0_pred
)
stackResults
```

```
##    knnN0 knnMM bayes vgam lda
## 1      B     E     C    C   B
## 2      A     A     C    A   A
## 3      A     A     C    B   B
## 4      A     A     C    C   C
## 5      A     A     B    A   C
## 6      C     A     A    E   E
## 7      D     D     C    D   D
## 8      D     B     D    E   D
## 9      A     A     A    A   A
## 10     A     A     C    A   A
## 11     D     B     C    C   D
## 12     C     A     C    A   A
## 13     B     B     B    B   B
```

```
## 14      A      A      A     A    A
## 15      E      E      E     E    E
## 16      E      E      B     E    A
## 17      A      E      C     A    A
## 18      B      B      B     B    B
## 19      B      B      D     B    B
## 20      B      B      B     B    B
```

7. Write Answers to File

```
answers = knnN0_TEST0_pred

pml_write_files = function(x){
  n = length(x)
  for(i in 1:n){
    filename = paste0("problem_id_",i,".txt")

write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FALSE)
  }
}

pml_write_files(answers)
Sys.time()

## [1] "2015-06-16 02:03:16 MST"
```