Machine Learning Project

Kyle Becker

April 4, 2017

Data

The data used for this project was provided by the group of research and development of groupware technologies (Ugulino, W.Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H.,2012). This dataset was created through recording the activities of 4 healthy individuals over 8 hours of study. The goal of this assignment is using machine learning algorithms predict which of the 5 activities each participant is performing based on measurements provided by accelerometers.

The dataset had all columns removed where NA was contained in all rows. Additionally certain columns existed in the training set but not in the testing set. Since these columns would not be useful in predicting in the test set (since they do not exist) they were removed from the training set. A summary of each variable is provided below.

```
str(Training_Data_Set)
```

```
## 'data.frame':    19622 obs. of  53 variables:
##  $ 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              : Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
```

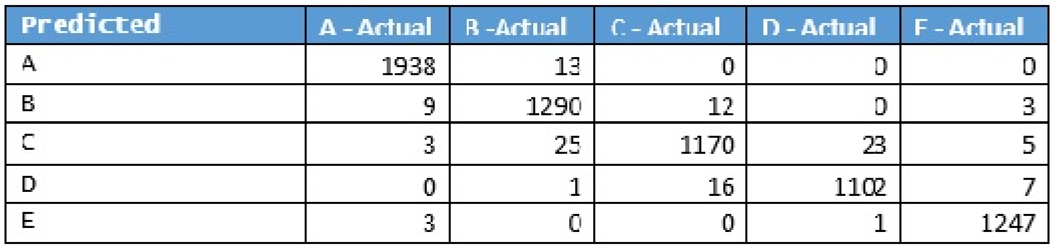
Once the data was preprocessed into the correct format a 30% sample size was taken for the training set. A 30% sample size was taken for processing reasons, normally a 70% training to 30% testing set would be taken but this resulted in too long of a processing time. The long processing time was due to the fact the computer being used for this analysis only had 8GB of ram.

```
library(caret)
library(stringi)

#Individual models

#Read data into R
Training_Data <- read.csv("C:\\Users\\kbec\\Desktop\\Machine Learning\\Assignment 4\\Training Da
ta.csv")
Testing_Data <- read.csv("C:\\Users\\kbec\\Desktop\\Machine Learning\\Assignment 4\\Testing Dat
a.csv")

#Remove first 7 columns because this is background info
Training_Data <- Training_Data[,8:160]
Testing_Data <- Testing_Data[,8:160]

# Remove columns were all rows are NA (not useful)
Testing_Remove_NA_Columns <- names(Testing_Data[,colSums(is.na(Testing_Data)) !=nrow(Testing_Dat
a)])

#Find columns that are the same between training and testing
Common_Columns <- subset(names(Training_Data), names(Training_Data) %in% Testing_Remove_NA_Colum
ns)

#Have Training and Testing only have columns that exist in both
#Include classe and problem id because these are the same column
Training_Data_Set <- Training_Data[, c(Common_Columns,"classe")]
Testing_Data_Set <- Testing_Data[,c(Common_Columns,"problem_id")]

#Ensure data sets have the same number of columns
dim(Testing_Data_Set)
dim(Training_Data_Set)

#Create Training set that includes 30%
#Create Testing set that includes 70% of data
#Training should be 70% but my computer cannot handle the analysis
Training_Partition <- createDataPartition(Training_Data_Set$classe, p=0.3, list = FALSE)
Testing <- Training_Data_Set[-Training_Partition,]
Training <- Training_Data_Set[Training_Partition,]
```

Analysis

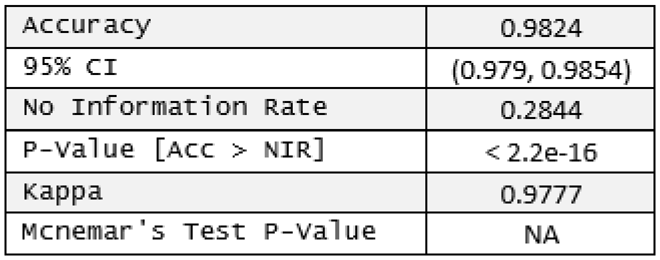
The 30% sample size was then ran through 3 different machine learning models to determine which was most effective in predicting the desired outcome (classe - the activity being performed). The 3 models used to predict classe were random forest, linear discriminant analysis and gradient boosting machine. On the initial run of each algorithm default inputs were used in conjunction with k fold cross validation. K fold cross validation to ensure that the models did not overfit and were generalizable to the test set. A value of 10 was used for K with 1 repeat meaning the 30% training set was split up into 10 groups which were separately trained and their results averaged. Once each model was trained using the training data the accuracy was measured against the remaining testing data to ensure the results were generalizable. The Results were measured using a confusion matrix which used accuracy of classification (predicted vs actual value of testing set). The results are displayed below. Using base parameters random forest was the most successful. The results are shown below.

Random Forest Confusion Matrix



Predicted	A - Actual	B -Actual	C - Actual	D - Actual	E - Actual
A	1938	13	0	0	0
B	9	1290	12	0	3
C	3	25	1170	23	5
D	0	1	16	1102	7
E	3	0	0	1	1247

Random For Model Accuracy



Accuracy	0.9824
95% CI	(0.979, 0.9854)
No Information Rate	0.2844
P-Value [Acc > NIR]	< 2.2e-16
Kappa	0.9777
Mcnemar's Test P-Value	NA

The confusion matrix shows that there was not a very large amount of misclassification which is represented in the model accuracy score of 98.24%. Additionally the confidence interval shows that 95% of the models created using cross validation fell within an accuracy score of 97.9% and 98.54%. The p value is less than .05 showing the results were statistically significant. The Kappa statistic of 97.7% is extremely promising because it takes into account the possibility of this accuracy being observed by chance.

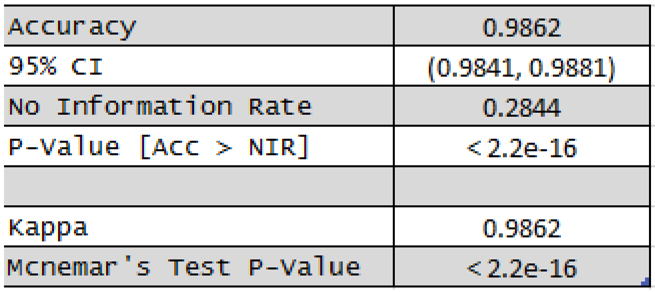
Further Analysis

In order to determine if other algorithms would be equally likely to achieve the same accuracy several other methods were tested. Specifically random forest, linear discriminant analysis and gradient boosting machines as well as with bootstrap aggregation with random forest and model stacking of random forest, linear discriminant analysis and gradient boosting together. The code for each model is shown below.

```
#Individual Model

#Using the caret package I apply the gradient boosting machine algorithm on the training data se
t
GBM_1 <- train(classe~., method="gbm", data = Training, verbose=FALSE)

#Using the caret package I apply the random forest algorithm on the training data set
RF_1 <- train(classe~., method="rf", data =Training, trControl= trainControl(method = "repeatedc
v", number = 10, repeats = 1))

#Using the caret package I apply the linear discriminant analysis algorithm on the training data
set
LDA_1 <- train(classe~., method="lda", data=Training, trControl= trainControl(method = "repeated
cv", number = 10, repeats = 1))

#Once each model has run I use each model to predict the classe for each record using the testin
g set
GBM_Predict_1 <- predict(GBM_1, Testing)
RF_Predict_1 <- predict(RF_1, Testing)
LDA_Predict_1 <- predict(LDA_1, Testing)

#After the predictions are finished a confusion matrix is made for each model displaying what th
e model predicted vs its actual classe in order to measure model accuracy
GBM_CM <- confusionMatrix(GBM_Predict_1, Testing$classe)
RF_CM <- confusionMatrix(RF_Predict_1, Testing$classe)
LDA_CM <- confusionMatrix(LDA_Predict_1, Testing$classe)

#Stacking Models

#When stacking models since they are ultimately ran individually and then combined a validation
 set is required to test the result of the stacked models

# 30% of the data is used for the validation set, 30% for the training set and 40% forr the test
ing because as mentioned there is an equipment limitation with the computer
Training_Partition <- createDataPartition(Training_Data_Set$classe, p=0.7, list = FALSE)
Validation <- Training_Data_Set[-Training_Partition,]
Non_validation <- Training_Data_Set[Training_Partition,]
Training_Testing_Partition <- createDataPartition(Non_validation$classe, p=0.3, list = FALSE)
Training <- Non_validation[Training_Testing_Partition,]
Testing <- Non_validation[-Training_Testing_Partition,]

#Each model is trained using default parameters
GBM_2 <- train(classe~., method="gbm", data = Training, verbose=FALSE, trControl= trainControl(m
ethod = "repeatedcv", number = 10, repeats = 1))

RF_2 <- train(classe~., method="rf", data =Training, trControl= trainControl(method = "repeatedc
v", number = 10, repeats = 1), VERBOSE=FALSE)

LDA_2 <- train(classe~., method="lda", data=Training, trControl= trainControl(method = "repeated
cv", number = 10, repeats = 1), VERBOSE=FALSE)

#Each model is used to create predicted values using the testing set
GBM_Predict_2 <- predict(GBM_2, Testing)
```

```
RF_Predict_2 <- predict(RF_2, Testing)
LDA_Predict_2 <- predict(LDA_2, Testing)



#Each prediction is combined into a dataframe
DF <- data.frame(GBM_Predict_2, RF_Predict_2, LDA_Predict_2, classe=Testing$classe)

#The dataframe is then ran through random forest using default parameters
RF_Combo <- train(classe~., method="rf", data = DF, trControl= trainControl(method = "repeatedc
v", number = 10, repeats = 1))

#The model is then tested using the validation set
#first the validation set is ran through each of the original models
GBM_Predict_v <- predict(GBM_2, Validation)
RF_Predict_v <- predict(RF_2, Validation)
LDA_Predict_v <- predict(LDA_2, Validation)

#Then the results are combined into a dataframe
DF <- data.frame(GBM_Predict_2=GBM_Predict_v, RF_Predict_2=RF_Predict_v, LDA_Predict_2=LDA_Predi
ct_v)

#This dataframe is then used to predict using the stacked model
RF_Combo_Predict <- predict(RF_Combo, DF)

#The predicted results are then compared to the actual results of the validation set to determin
e accuracy
RF_Combo_CM <- confusionMatrix(RF_Combo_Predict, Validation$classe)
RF_Combo_CM

#Bagging

#The bagging library takes the predictors and output variable as different arguments so they are
split out into different dataframes
predictors <- Training[1:52]
y <- Training$classe

#Using the bag function these dataframes are passed followed by B which states the number of boo
tsrap aggregation samples

#the model is fit using the fit function, they are predicted using the predict function and the
 results are aggregate using the aggregation function
treebag <- bag(predictors, y, B=10, bagControl = bagControl(fit = ctreeBag$fit, predict = ctreeB
ag$pred, aggregate = ctreeBag$aggregate))

#The model is then used to predict classes for the Testing data set
treebag_predict <- predict(treebag, Testing)

#A confusion matrix is then created comparing the predicted value vs the actual value to determi
ne accuracy
treebag_CM <- confusionMatrix(treebag_predict, Testing$classe)
treebag
```

Gradient Boosting Machine Accuracy



Accuracy	0.9862
95% CI	(0.9841, 0.9881)
No Information Rate	0.2844
P-Value [Acc > NIR]	< 2.2e-16
Kappa	0.9862
Mcnemar's Test P-Value	< 2.2e-16

The gradient boosting machine individually was very close to random forest producing an accuracy level on the test set of 95.52% using default parameters. The kappa was also very high at 94.33%. The parameters that alter the results of the algorithm are shrinkage, number of trees, interaction depth and nminobsinnode. The shrinkage parameter determines how quickly the algorithm learns, a high value reduces computational requirements but is prone to overfitting, the number of trees allows the algorithm to determine more complicated relationships if the number of trees is larger. If the shrinkage rate is low the number of trees will have to be increased since the algorithm learns slower. Interaction depth is the number of splits performed on a tree once again if this is increased computationally it is taxing but it will determine more complex relationships. Lastly nminobsinnode is the minimum number of observations required in the terminal node. In order to optimize the model a tuning grid was created which decreased the learning rate lower than the default 0.1 and increased the number of trees created. The remaining two parameters were held constant at a default. The final model arrived at the default learning rate (shrinkage) of 0.1 but used 500 trees rather than the default 150. This allowed the model to reach an accuracy level of 98.62% on the testing set higher than random forest. Out of the 10 cross fold validation the accuracy score fell with a 95% confidence interval of 98.41% and 98.81% once again higher than random forest. The P value was less than 0.05 showing the results were statistically significant. Lastly the most important statistic Kappa value was also higher than random forest at 98.26%. This is most important because as previously mentioned this takes into account the probability of the algorithm arriving at the results by random chance. THe expected out of sample error rate is therefore less than 1% 1-accuracy. The code is shown below.

```r
#GBM_GRID creates a list of parameters to be tested in the model creation process
GBM_Grid <- expand.grid(interaction.depth = 6, n.trees=c(100,300,500), shrinkage = seq(.01,.1,.0
3), n.minobsinnode = 10)

#These paremeters  are then passed to the tuneGrid parameter to be tested in the GBM Model
GBM_Optimize <-  train(classe~., method="gbm", data = Training, verbose=FALSE, trControl= trainC
ontrol(method = "repeatedcv", number = 10, repeats = 1), tuneGrid=GBM_Grid)

#The model is then used on the testing set to predict the classe of each record
GBM_Predict_Optimize <- predict(GBM_Optimize, Testing)

#A confusion matrix is then used to determine the accuracy using predicted vs actual value
GBM_Optimize_CM <- confusionMatrix(GBM_Predict_Optimize, Testing$classe)
```

Linear discriminant analysis was not pursued further because it did not perform well compared to the other models at 70.38% and a Kappa value of 62.56%. The decision tree model through bootstrap aggregation was somewhat effective producing an accuracy of 92.58% and a Kappa score of 90.60% . Lastly stacking the models through random forest was almost as effective as the optimized parameters of the gradient boosting machine producing an accuracy of 98.39% and a Kappa value of 97.96%. However, this algorithm if deployed would be computationally much more taxing so the optimized gradient boosting machine model was chosen as the most efficient classifier.