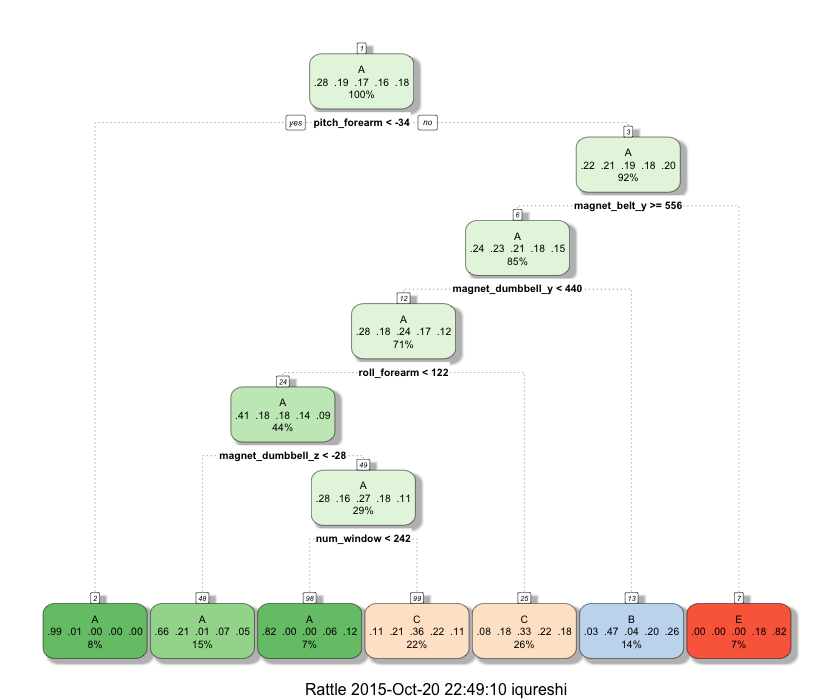
* Data Cleansing:
* Before we are able to use the data set provided on our algorithm, we need to clean the data. The data contained several columns with a value of “#DIV!”. So while reading the data from file, I turned “#DIV!” into NA. So the total NA values include cell values with “NA” as well as cell values with “#DIV!”.
* data <- read.csv(“pml-training.csv”, header = TRUE, na.string = c(“#DIV!”, “NA”, stringsAsFactors = FALSE)
* Quick visual glance at the data tells us that first 6 columns are not contributing to predict “classe”. They were time variables, user name. So out of 160 variables read, I removed the first 6 variables as part of cleansing.
* data <- data <- [, -(1:6)]
* Another thing that stands out by visual glance at the data is that there are some columns where most values equal to “NA”. So, to take care of that, I removed any columns from our dataset that have more than 80% “NA” values.
* cols <- apply (data, 2, function(x) sum(is.na(x)))
* data\_na <- data[, cols < (nrow(data) \* .2)]
* This removed another hundred columns from our data and we are looking at a reasonable 54 columns instead of original 160. But of course we can do better. Now, I looked at highly correlated predictors and eliminated anything that has a correlation greater than 90%. This really can be reduced to even 70-80% but I wanted to make sure that I don’t compromise on accuracy.
* highlyCorPred <- findCorrelation(cor(data\_na[, -54]), cutoff = 0.9)
* data\_na <- data\_na [, -highlyCorPred]
* This leaves me with only 41 variables to use as predictors.
* At this point I thought about using PCA to further reduce the predictors but using PCA means I will lose the ability to infer. Given dataset is only 12MB, and 41 predictors is not a terribly bad number even if you have a laptop, so I will keep my 41 variables and not use PCA. A quick test on trying to use PCA tells me that 20 variables will retain 90% variability.
* Now, I am ready to create my training, cross validation and test set.
* inTrain <- createDataPartition (y=data\_na$classe, p=0.7, list=FALSE, t=2)
* training <- data\_na [, inTrain]
* testing <- data\_na[, -inTrain]
* x\_validation <- data\_na[-inTrain[, 2], ] #notice t = 2 in createDataPartition
* Choosing algorithm:
* All datasets have been prepared and data have been cleaned. Now I have to choose an algorithm.
* I initially tried to use linear algorithm rather than using a tree algorithm. Very soon I learned the glm cannot be used to predict multiple classes. In other words, the problem we have is a multi class classification which glm canbe used to solve binary classification. So that goes out the door.
* Then I was down to using tree algorithms. I could create my model using rpart, random forrest or bagging (tree bag).
* On first run of trying to create a model using rpart, I got an error on classe variable. It has been read as “chr”. This needs to be Factor. So, I converted classe to factor.
* training$classe <- as.factor (training$classe)
* now run the model using rpart
* modelFit <- train (classe ~ ., method = "rpart", data=training)
* 
* Looking at this tree model we can easily traverse and see that our predictors that are really contributing to predict classe variable are six. They are:
* Pitch\_forearm
* Magnet\_belt\_y
* Magnet\_dumbbell\_y
* Role\_forearm
* Magnet\_dumbbell\_z
* num\_window
* This unfortunately is not very accurate. Here are the results for rpart
* > modelFit$results
* cp Accuracy Kappa AccuracySD KappaSD
* 1 0.03473706 0.5470362 0.4265475 0.01535966 0.01887568
* 2 0.03867867 0.5113930 0.3726988 0.03362248 0.05275437
* 3 0.06602838 0.3969048 0.1859103 0.09391434 0.15495164
* As you can see, Accuracy is horrible. So this is not the model, I want to use. Next I want to try bagging to see if it improves accuracy.
* For this purpose I choose treebag.
* > modelFit <- train (classe ~ ., method = "treebag", data=training)
* > modelFit$results
* parameter Accuracy Kappa AccuracySD KappaSD

1. none 0.9941846 0.9926424 0.001983685 0.002510777

This model is pretty accurate, to the point that I think this is over fitting my data. Being a novice, I am not sure if a model without over fitting can be this good.

So, I want to try other methods like gbm.

I create another model based on “gbm” and here is what I get.

gbm\_modelFit <- train(classe~ ., method = "gbm", data=training, verbose = FALSE)

* > gbm\_modelFit$results
* shrinkage interaction.depth n.minobsinnode n.trees Accuracy Kappa AccuracySD KappaSD
* 1 0.1 1 10 50 0.7362443 0.6652206 0.005829953 0.007295411
* 4 0.1 2 10 50 0.8744365 0.8409511 0.004568948 0.005805287
* 7 0.1 3 10 50 0.9323927 0.9143904 0.005082109 0.006464350
* 2 0.1 1 10 100 0.8097637 0.7590058 0.004130229 0.005157751
* 5 0.1 2 10 100 0.9397778 0.9237704 0.003868634 0.004899705
* 8 0.1 3 10 100 0.9764632 0.9702163 0.002602426 0.003298157
* 3 0.1 1 10 150 0.8540641 0.8152253 0.003417016 0.004296838
* 6 0.1 2 10 150 0.9670954 0.9583618 0.003075631 0.003896764

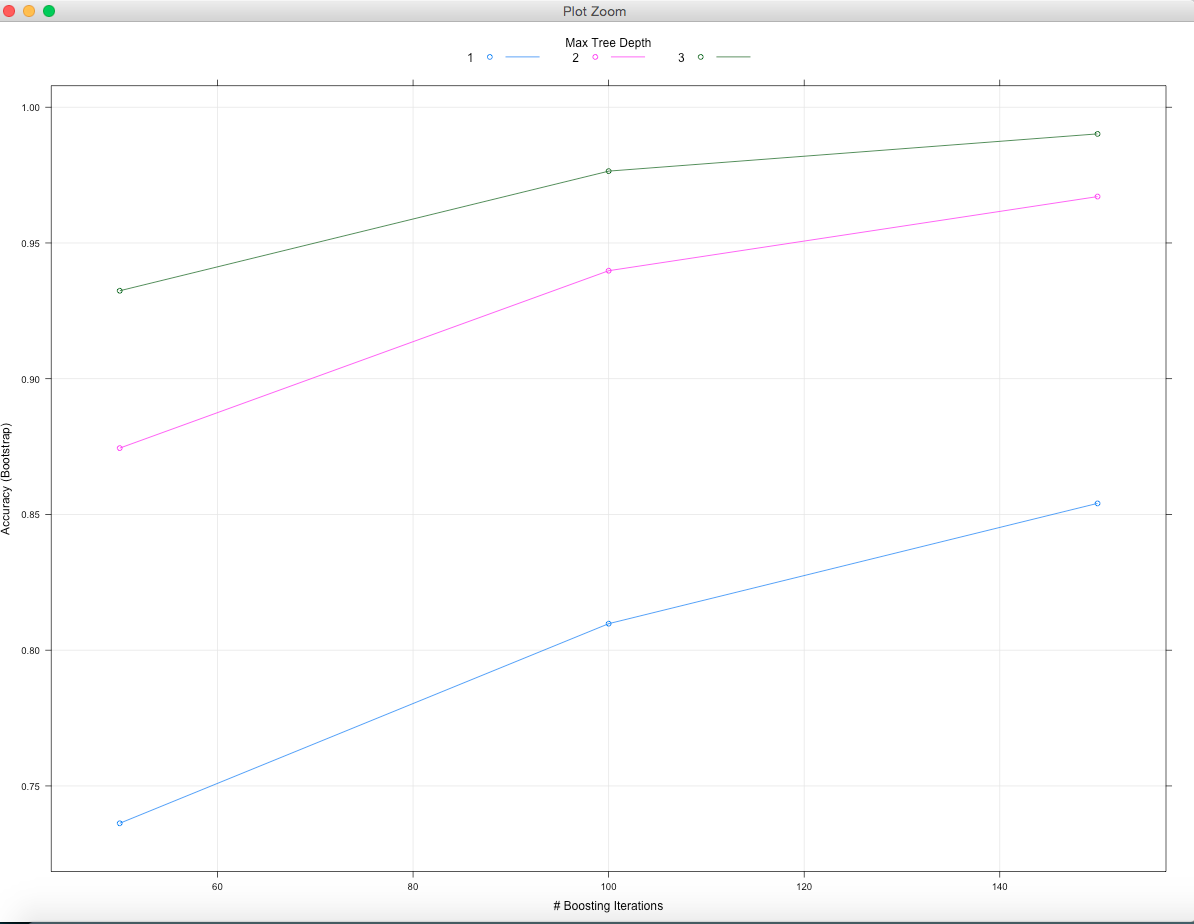
9 0.1 3 10 150 0.9901671 0.9875584 0.001504606 0.001906313

I run this gbm model on my x-validation set and results are pretty encouraging.

> prediction <- predict(gbm\_modelFit, x\_validation\_set)

> confusionMatrix(prediction)

Confusion matrix of this model tells us that its pretty accurate.

* Statistics by Class:
* Class: A Class: B Class: C Class: D Class: E
* Sensitivity 0.9970 0.9886 0.9864 0.9865 0.9945
* Specificity 0.9981 0.9966 0.9975 0.9978 0.9992
* Pos Pred Value 0.9952 0.9860 0.9883 0.9886 0.9963
* Neg Pred Value 0.9988 0.9973 0.9971 0.9974 0.9988
* Prevalence 0.2845 0.1935 0.1743 0.1638 0.1839
* Detection Rate 0.2836 0.1913 0.1720 0.1616 0.1828
* Detection Prevalence 0.2850 0.1941 0.1740 0.1635 0.1835
* Balanced Accuracy 0.9976 0.9926 0.9919 0.9921 0.9968
* > gbm\_modelFit
* Stochastic Gradient Boosting
* 27474 samples
* 41 predictor
* 5 classes: 'A', 'B', 'C', 'D', 'E'
* No pre-processing
* Resampling: Bootstrapped (25 reps)
* Summary of sample sizes: 27474, 27474, 27474, 27474, 27474, 27474, ...
* Resampling results across tuning parameters:
* interaction.depth n.trees Accuracy Kappa Accuracy SD Kappa SD
* 1 50 0.7362443 0.6652206 0.005829953 0.007295411
* 1 100 0.8097637 0.7590058 0.004130229 0.005157751
* 1 150 0.8540641 0.8152253 0.003417016 0.004296838
* 2 50 0.8744365 0.8409511 0.004568948 0.005805287
* 2 100 0.9397778 0.9237704 0.003868634 0.004899705
* 2 150 0.9670954 0.9583618 0.003075631 0.003896764
* 3 50 0.9323927 0.9143904 0.005082109 0.006464350
* 3 100 0.9764632 0.9702163 0.002602426 0.003298157
* 3 150 0.9901671 0.9875584 0.001504606 0.001906313
* Tuning parameter 'shrinkage' was held constant at a value of 0.1
* Tuning parameter 'n.minobsinnode' was
* held constant at a value of 10
* Accuracy was used to select the optimal model using the largest value.
* The final values used for the model were n.trees = 150, interaction.depth = 3, shrinkage = 0.1
* and n.minobsinnode = 10.
* The best tune option for this model is
* >gbm\_modelFit$bestTune
* n.trees interaction.depth shrinkage n.minobsinnode
* 9 150 3 0.1 10
* Another important aspect to look at is the finalModel object. This will tell us how many variables actually contributed.
* >gbm\_modelFit$finalModel
* A gradient boosted model with multinomial loss function. 150 iterations were performed. There were 41 predictors of which 33 had non-zero influence.
* Plotting the model to see the estimates of performance and tuning parameters.
* 
* As you can see, Accuracy increases as we increase number of iterations and tree depth.
* At this point, I can use either gbm or treebag. I continue with treebag.
* In Sample and Out of Sample Error:
* To check In Sample Error, run predict and then show confusion matrix on training data.
* > confusionMatrix(train\_predict, training$classe)
* Confusion Matrix and Statistics
* Reference
* Prediction A B C D E
* A 7804 23 0 0 0
* B 6 5274 36 5 0
* C 0 9 4748 37 2
* D 2 10 8 4460 19
* E 0 0 0 2 5029
* Overall Statistics
* Accuracy : 0.9942
* 95% CI : (0.9932, 0.9951)
* Accuracy in case of in sample is 99.42%.
* To check out of sample error, I ran predict on cross validation and then call the confusion matrix to see out of sample error.
* > prediction <- predict(modelFit, x\_validation\_set)
* > confusionMatrix(prediction, x\_validation\_set$classe)
* Confusion Matrix and Statistics
* Reference
* Prediction A B C D E
* A 1669 8 0 0 0
* B 4 1126 12 0 0
* C 0 2 1012 10 0
* D 1 2 2 951 6
* E 0 1 0 3 1076
* Overall Statistics
* Accuracy : 0.9913
* 95% CI : (0.9886, 0.9935)
* From both confusion matrices, we can see that in sample error is lower than out of sample error. Accuracy is high.
* Finally I ran the model on test data created using “CreateDataPartition()” earlier and the results are very encouraging. The out of sample error is little more than train error as expected but accuracy of algorithm is still pretty high.
* > confusionMatrix(test\_predict, testing$classe)
* Confusion Matrix and Statistics
* Reference
* Prediction A B C D E
* A 481 3 0 0 0
* B 0 343 2 0 0
* C 0 0 310 5 0
* D 0 1 1 277 2
* E 0 1 0 1 344
* Overall Statistics
* Accuracy : 0.991
* Variable of Importance:
* To find which variables are contributing most towards prediction, I ran varImp() method on my model. Please see the result.
* > varImp(modelFit)
* treebag variable importance
* only 20 most important variables shown (out of 41)
* Overall
* num\_window 100.00
* yaw\_belt 74.09
* magnet\_dumbbell\_y 46.49
* magnet\_dumbbell\_z 43.54
* accel\_dumbbell\_y 37.31
* roll\_forearm 34.70
* magnet\_belt\_y 34.32
* pitch\_forearm 33.18
* total\_accel\_belt 29.30
* gyros\_belt\_z 29.02
* roll\_dumbbell 27.39
* magnet\_dumbbell\_x 24.19
* magnet\_belt\_z 22.53
* magnet\_forearm\_z 18.86
* total\_accel\_dumbbell 17.55
* magnet\_belt\_x 15.46
* roll\_arm 14.91
* magnet\_arm\_x 14.40
* accel\_forearm\_x 13.14
* yaw\_dumbbell 12.33
* We can see 20 most important variables that are contributing towards the outcome. Knowing these, we can have our users improve their exercise so they are doing it right.