Coursera Machine Learning Project - Barbell Lifts Classe Prediction

Betsy Nash

March 11, 2018

Synopsis

The data from 6 participants is available to see if a prediction model can determine if the manner in which they did the exercise. This is the “classe” variable in the dataset. There are 5 classes: sitting-down, standing-up, standing, walking, and sitting.

Three prediction models were used in this review. The random forest model is determined to be the best based solely on an accuracy measure, as determined by a parsed test dataset.

The data is provided from Proceedings of 21st Brazilian Symposium on Artificial Intelligence, Ugulino, W., et al, Wearable Computing: Accelerometers’ Data Classification of Body Postures and Movements.

Loaded Libraries

```
library(dplyr)
library(ggplot2)
library(caret)
library(rattle)
library(rpart)
library(rpart.plot)
library(randomForest)
```

Clean & Tidy Dataset

The training dataset is found here:

```
Connection<-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
TrainDestFile<-"pml-training.csv"
download.file(Connection,TrainDestFile)
TrainData<- read.csv("pml-training.csv")
#in the interest of document length, not displaying the structure
#str(TrainData)
```

The following observations were made looking at the structure: many NAs & #DIV/0! error. Blanks are also a concern. Prediction algorithms fail with missing values. Therefore, create a dataset with a treatment for missing values.

```
TrainDataUse<- read.csv("pml-training.csv",na.strings=c("NA","#DIV/0!",""))
#check dimension unchanged
#str(TrainDataUse)
```

With the treatment in place, now we need to identify columns with a significant amount of NA’s. A threshold of 50% is used (essentially a coin flip whether there is useable data or not). Any column with more than 50% NA’s is removed from analysis.

```
#count of NAs by col using simple apply
#In the interest of report length, it is not displayed, but here is the code
#sapply(TrainDataUse, function(x) sum(is.na(x)))
#identify cols with > 50% NAs
indColToRemove <- which(colSums(is.na(TrainDataUse))>0.5*dim(TrainDataUse)[1])
print(indColToRemove)
```

```
##       kurtosis_roll_belt      kurtosis_picth_belt        kurtosis_yaw_belt
##                       12                       13                       14
##       skewness_roll_belt     skewness_roll_belt.1        skewness_yaw_belt
##                       15                       16                       17
##            max_roll_belt           max_picth_belt             max_yaw_belt
##                       18                       19                       20
##            min_roll_belt           min_pitch_belt             min_yaw_belt
##                       21                       22                       23
##      amplitude_roll_belt     amplitude_pitch_belt       amplitude_yaw_belt
##                       24                       25                       26
##     var_total_accel_belt            avg_roll_belt         stddev_roll_belt
##                       27                       28                       29
##            var_roll_belt           avg_pitch_belt        stddev_pitch_belt
##                       30                       31                       32
##           var_pitch_belt             avg_yaw_belt          stddev_yaw_belt
##                       33                       34                       35
##             var_yaw_belt            var_accel_arm             avg_roll_arm
##                       36                       50                       51
##           stddev_roll_arm             var_roll_arm            avg_pitch_arm
##                       52                       53                       54
##          stddev_pitch_arm            var_pitch_arm              avg_yaw_arm
##                       55                       56                       57
##           stddev_yaw_arm              var_yaw_arm        kurtosis_roll_arm
##                       58                       59                       69
##       kurtosis_picth_arm         kurtosis_yaw_arm        skewness_roll_arm
##                       70                       71                       72
##       skewness_pitch_arm         skewness_yaw_arm             max_roll_arm
##                       73                       74                       75
##            max_picth_arm              max_yaw_arm             min_roll_arm
##                       76                       77                       78
##            min_pitch_arm              min_yaw_arm       amplitude_roll_arm
##                       79                       80                       81
##      amplitude_pitch_arm        amplitude_yaw_arm   kurtosis_roll_dumbbell
##                       82                       83                       87
##  kurtosis_picth_dumbbell    kurtosis_yaw_dumbbell   skewness_roll_dumbbell
##                       88                       89                       90
##  skewness_pitch_dumbbell    skewness_yaw_dumbbell        max_roll_dumbbell
##                       91                       92                       93
##       max_picth_dumbbell         max_yaw_dumbbell        min_roll_dumbbell
##                       94                       95                       96
##       min_pitch_dumbbell         min_yaw_dumbbell  amplitude_roll_dumbbell
##                       97                       98                       99
## amplitude_pitch_dumbbell   amplitude_yaw_dumbbell       var_accel_dumbbell
##                      100                      101                      103
##        avg_roll_dumbbell     stddev_roll_dumbbell        var_roll_dumbbell
##                      104                      105                      106
##       avg_pitch_dumbbell    stddev_pitch_dumbbell       var_pitch_dumbbell
##                      107                      108                      109
```

```
##         avg_yaw_dumbbell      stddev_yaw_dumbbell         var_yaw_dumbbell
##                      110                      111                      112
##    kurtosis_roll_forearm   kurtosis_picth_forearm     kurtosis_yaw_forearm
##                      125                      126                      127
##    skewness_roll_forearm   skewness_pitch_forearm     skewness_yaw_forearm
##                      128                      129                      130
##         max_roll_forearm        max_picth_forearm          max_yaw_forearm
##                      131                      132                      133
##         min_roll_forearm        min_pitch_forearm          min_yaw_forearm
##                      134                      135                      136
##   amplitude_roll_forearm  amplitude_pitch_forearm    amplitude_yaw_forearm
##                      137                      138                      139
##        var_accel_forearm         avg_roll_forearm      stddev_roll_forearm
##                      141                      142                      143
##         var_roll_forearm        avg_pitch_forearm     stddev_pitch_forearm
##                      144                      145                      146
##        var_pitch_forearm          avg_yaw_forearm       stddev_yaw_forearm
##                      147                      148                      149
##          var_yaw_forearm
##                      150
```

```
#count = 100, now remove these columns
TrainDataUseClean <- TrainDataUse[,-indColToRemove]
#in the interest of report length, while not displayed, the next two lines were used t
o confirm 100 columns are removed
#dim(TrainDataUseClean)
#dim(TrainDataUse)
```

Next the data was reviewed for any obvious columns that would not be good predictors. Two columns were removed from the dataset.

```
#In the interest of report lenth, not displaying head, but it was used to determine wh
ich columns to trim.
#head(TrainDataUseClean,10)
#Trim: do not need cols X (rowID) or user_name. They may effect the prediction model.
TrainDataUseCleanTrim <- TrainDataUseClean[,-c(1:2)]
#While not displayed, the following line was used to confirm the new column count 60
-2 = 58
#dim(TrainDataUseCleanTrim)
```

A check for the outcome column, classe, is performed to make sure there are no holes/gaps to address.

```
UniqueClasse<-distinct(select(TrainDataUseCleanTrim, classe))
#In the interest of report length, not showing the results. No issues identified.
#print(UniqueClasse)
```

With the cleansing done on the training set, the next step is to explore the data for any additional adjustments. Summary and near zero variance are reviewed.

```
#In the interest of report length, not displaying summary. No issues found.
#summary(TrainDataUseCleanTrim)
#Look for near zero variance columns
nsv<-nearZeroVar(TrainDataUseCleanTrim,saveMetrics=TRUE)
nsv
```

```
##                      freqRatio percentUnique zeroVar   nzv
## raw_timestamp_part_1  1.000000    4.26562022   FALSE FALSE
## raw_timestamp_part_2  1.000000   85.53154622   FALSE FALSE
## cvtd_timestamp        1.000668    0.10192641   FALSE FALSE
## new_window           47.330049    0.01019264   FALSE  TRUE
## num_window            1.000000    4.37264295   FALSE FALSE
## roll_belt             1.101904    6.77810621   FALSE FALSE
## pitch_belt            1.036082    9.37722964   FALSE FALSE
## yaw_belt              1.058480    9.97349913   FALSE FALSE
## total_accel_belt      1.063160    0.14779329   FALSE FALSE
## gyros_belt_x          1.058651    0.71348486   FALSE FALSE
## gyros_belt_y          1.144000    0.35164611   FALSE FALSE
## gyros_belt_z          1.066214    0.86127816   FALSE FALSE
## accel_belt_x          1.055412    0.83579655   FALSE FALSE
## accel_belt_y          1.113725    0.72877383   FALSE FALSE
## accel_belt_z          1.078767    1.52379982   FALSE FALSE
## magnet_belt_x         1.090141    1.66649679   FALSE FALSE
## magnet_belt_y         1.099688    1.51870350   FALSE FALSE
## magnet_belt_z         1.006369    2.32901845   FALSE FALSE
## roll_arm             52.338462   13.52563449   FALSE FALSE
## pitch_arm            87.256410   15.73234125   FALSE FALSE
## yaw_arm              33.029126   14.65701763   FALSE FALSE
## total_accel_arm       1.024526    0.33635715   FALSE FALSE
## gyros_arm_x           1.015504    3.27693405   FALSE FALSE
## gyros_arm_y           1.454369    1.91621649   FALSE FALSE
## gyros_arm_z           1.110687    1.26388747   FALSE FALSE
## accel_arm_x           1.017341    3.95984099   FALSE FALSE
## accel_arm_y           1.140187    2.73672409   FALSE FALSE
## accel_arm_z           1.128000    4.03628580   FALSE FALSE
## magnet_arm_x          1.000000    6.82397309   FALSE FALSE
## magnet_arm_y          1.056818    4.44399144   FALSE FALSE
## magnet_arm_z          1.036364    6.44684538   FALSE FALSE
## roll_dumbbell         1.022388   84.20650290   FALSE FALSE
## pitch_dumbbell        2.277372   81.74498012   FALSE FALSE
## yaw_dumbbell          1.132231   83.48282540   FALSE FALSE
## total_accel_dumbbell  1.072634    0.21914178   FALSE FALSE
## gyros_dumbbell_x      1.003268    1.22821323   FALSE FALSE
## gyros_dumbbell_y      1.264957    1.41677709   FALSE FALSE
## gyros_dumbbell_z      1.060100    1.04984201   FALSE FALSE
## accel_dumbbell_x      1.018018    2.16593619   FALSE FALSE
## accel_dumbbell_y      1.053061    2.37488533   FALSE FALSE
## accel_dumbbell_z      1.133333    2.08949139   FALSE FALSE
## magnet_dumbbell_x     1.098266    5.74864948   FALSE FALSE
## magnet_dumbbell_y     1.197740    4.30129447   FALSE FALSE
## magnet_dumbbell_z     1.020833    3.44511263   FALSE FALSE
## roll_forearm         11.589286   11.08959331   FALSE FALSE
## pitch_forearm        65.983051   14.85577413   FALSE FALSE
## yaw_forearm          15.322835   10.14677403   FALSE FALSE
```

```
## total_accel_forearm   1.128928    0.35674243   FALSE FALSE
## gyros_forearm_x       1.059273    1.51870350   FALSE FALSE
## gyros_forearm_y       1.036554    3.77637346   FALSE FALSE
## gyros_forearm_z       1.122917    1.56457038   FALSE FALSE
## accel_forearm_x       1.126437    4.04647844   FALSE FALSE
## accel_forearm_y       1.059406    5.11160942   FALSE FALSE
## accel_forearm_z       1.006250    2.95586586   FALSE FALSE
## magnet_forearm_x      1.012346    7.76679238   FALSE FALSE
## magnet_forearm_y      1.246914    9.54031189   FALSE FALSE
## magnet_forearm_z      1.000000    8.57710733   FALSE FALSE
## classe                1.469581    0.02548160   FALSE FALSE
```

The “new_window” column is TRUE for near zero variance and needs to be removed as a possible covariate.

```
TrainDataUseCleanTrimCOV<-subset(TrainDataUseCleanTrim,select=-c(new_window))
```

For cleansing, four steps were performed to have a clean & tidy dataset for modeling:
1) identify what is considered a missing variable
2) remove columns with more than 50% NAs
3) removed the RowID and user_name columns
4) removed the new_window column due to near zero variance

Parsing the Dataset into Build and Test Datasets

A 75/25 split is used on the clean and tidy dataset. 75% for building the model. 25% for testing and determining accuracy of the models.

```
#set seed to make sure same random sample each iteration
set.seed(12345)
InBuild <- createDataPartition(TrainDataUseCleanTrimCOV$classe, p=0.75, list=FALSE)
BuildSet <- TrainDataUseCleanTrimCOV[InBuild,]
TestSet <- TrainDataUseCleanTrimCOV[-InBuild,]
#while not displayed, the following was used to confirm the split balances to the tota
l row count
#dim(BuildSet)
#dim(TestSet)
```

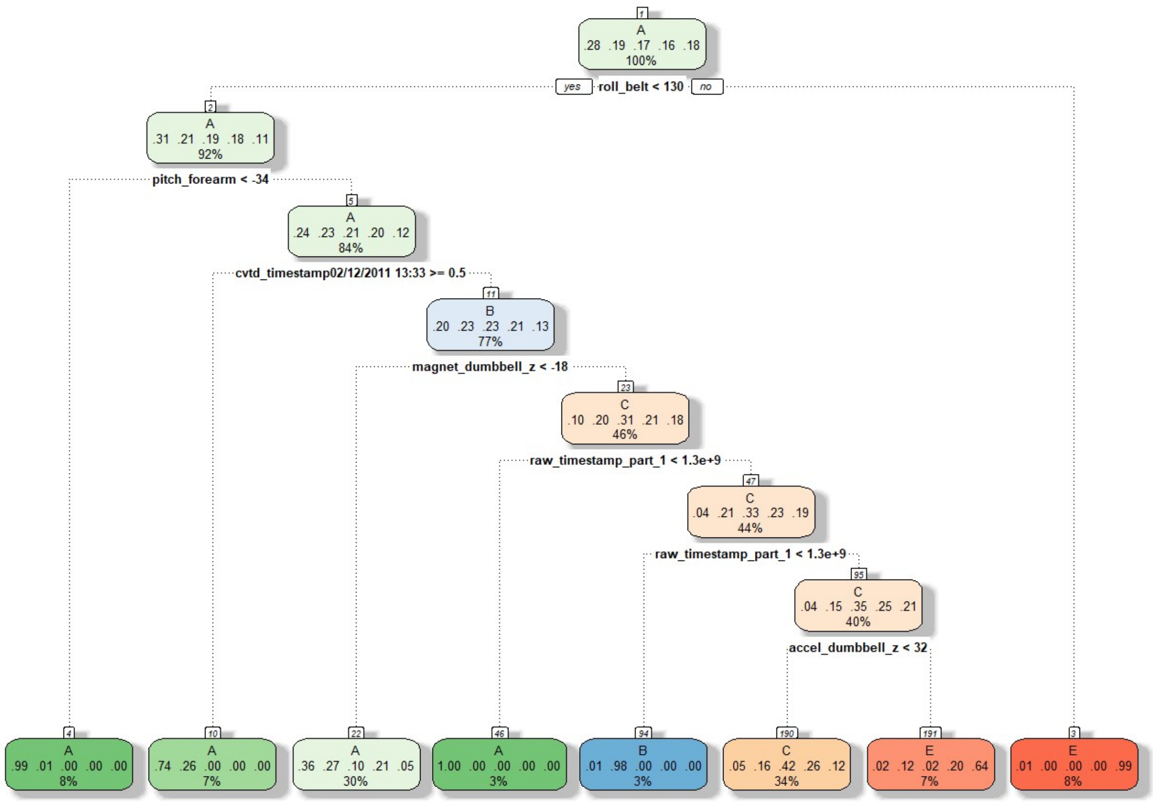
Machine Learning with the Caret Package

The goal of a predictive model is to find the signal (good predictor variables). Three models were explored in this review. Cross-validation involves a process of fitting models and testing them.

1. Decision Tree
2. Random Forest
3. Gradient Boosting

The best model is selected in this review based solely on the measure of accuracy on the test dataset.

```
#set cross validation and folds = 5
Cntl<-trainControl(method="cv",number = 5)
FitTree<-train(classe ~ ., data=BuildSet, method="rpart", trControl=Cntl)
#view final model
fancyRpartPlot(FitTree$finalModel)
```



Rattle 2018-Mar-11 18:34:56 Betsy

```
#predicting new values with test set
TreePredict<-predict(FitTree,newdata=TestSet)
#compare classe with the model using confusion matrix to see if it is a good fit or no
t
confusionMatrix(TreePredict,TestSet$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1308  507  157  268   88
##          B    4  132    0    0    0
##          C   73  274  696  442  217
##          D    0    0    0    0    0
##          E   10   36    2   94  596
##
## Overall Statistics
##
##                Accuracy : 0.5571
##                  95% CI : (0.5431, 0.5711)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4259
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9376  0.13909   0.8140   0.0000   0.6615
## Specificity            0.7093  0.99899   0.7515   1.0000   0.9645
## Pos Pred Value         0.5619  0.97059   0.4089      NaN   0.8076
## Neg Pred Value         0.9662  0.82865   0.9503   0.8361   0.9268
## Prevalence             0.2845  0.19352   0.1743   0.1639   0.1837
## Detection Rate         0.2667  0.02692   0.1419   0.0000   0.1215
## Detection Prevalence   0.4747  0.02773   0.3471   0.0000   0.1505
## Balanced Accuracy      0.8235  0.56904   0.7828   0.5000   0.8130
```

Decision Tree: The accuracy on the test dataset is low. The next step is to explore the random forest model.

```
#same controls as that used in the decision tree
FitRF<-train(classe ~ ., data=BuildSet, method="rf", trControl = Cntl, verbose = FALS
E)
print(FitRF)
```

```
## Random Forest
##
## 14718 samples
##    56 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 11776, 11775, 11773, 11774, 11774
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9919824  0.9898573
##   38    0.9989129  0.9986249
##   74    0.9980977  0.9975939
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 38.
```

```
#predicting new values using best tree with test set
RFPredict<-predict(FitRF,newdata=TestSet)
#compare classe with the model using confusion matrix to see if it is a good fit or no
t
confusionMatrix(RFPredict,TestSet$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    1    0    0    0
##          B    0  947    1    0    0
##          C    0    1  854    0    0
##          D    0    0    0  804    0
##          E    0    0    0    0  901
## 
## Overall Statistics
##                                           
##                Accuracy : 0.9994          
##                  95% CI : (0.9982, 0.9999)
##     No Information Rate : 0.2845          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.9992          
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9979   0.9988   1.0000   1.0000
## Specificity            0.9997   0.9997   0.9998   1.0000   1.0000
## Pos Pred Value         0.9993   0.9989   0.9988   1.0000   1.0000
## Neg Pred Value         1.0000   0.9995   0.9998   1.0000   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1931   0.1741   0.1639   0.1837
## Detection Prevalence   0.2847   0.1933   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9999   0.9988   0.9993   1.0000   1.0000
```

```
CMRF<-confusionMatrix(RFPredict,TestSet$classe)
```

Random Forest: The accuracy on the test dataseet is very high, near 100%. This is a very promising model. The next step is to explore the gradient boosting method.

```
#same controls as earlier
FitGBM<-train(classe~., data=BuildSet, method="gbm", trControl=Cntl, verbose=FALSE)
print(FitGBM)
```

```
## Stochastic Gradient Boosting
##
## 14718 samples
##    56 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 11773, 11775, 11775, 11775, 11774
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.8448144  0.8030947
##   1                  100      0.8965887  0.8690135
##   1                  150      0.9257365  0.9059255
##   2                   50      0.9574665  0.9461544
##   2                  100      0.9871582  0.9837580
##   2                  150      0.9921184  0.9900311
##   3                   50      0.9853236  0.9814359
##   3                  100      0.9930015  0.9911481
##   3                  150      0.9961950  0.9951873
##
## 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.
```

```
GBMPredict<-predict(FitGBM,newdata=TestSet)
confusionMatrix(GBMPredict,TestSet$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    4    0    0    0
##          B    0  942    1    0    0
##          C    0    3  849    1    0
##          D    0    0    5  802    2
##          E    0    0    0    1  899
##
## Overall Statistics
##
##                Accuracy : 0.9965
##                  95% CI : (0.9945, 0.998)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9956
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9926   0.9930   0.9975   0.9978
## Specificity            0.9989   0.9997   0.9990   0.9983   0.9998
## Pos Pred Value         0.9971   0.9989   0.9953   0.9913   0.9989
## Neg Pred Value         1.0000   0.9982   0.9985   0.9995   0.9995
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1921   0.1731   0.1635   0.1833
## Detection Prevalence   0.2853   0.1923   0.1739   0.1650   0.1835
## Balanced Accuracy      0.9994   0.9962   0.9960   0.9979   0.9988
```

Gradient Boosting: The accuracy on the test dataset is also very high, but not as high as random forest. Therefore, the random forest model is the one selected for the prediction model.

The estimated out-of-sample error should be greater than the in-sample-error. The prediction model tunes a little bit to the noice in the build dataset. The test dataset will have different noise and the accuracy will lower a bit. The more realistic expectation is the performance of the model on the test dataset. The estimated out-of-sample error using our selected random forest model is 0.99%.

Interpretation of Results

Based strictly on accuracy, random forest model is the best prediction algorithm. The next step is to apply it to the validation dataset provided in the assignment. The same steps for cleaning and tidying the data in the training datset need to be applied to the validation dataset.

```
Connection2<-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
ValDestFile<-"pml-testing.csv"
download.file(Connection2,ValDestFile)
ValData<- read.csv("pml-testing.csv")
dim(ValData)
```

```
## [1]  20 160
```

```
#step 1 - tell R what to consider as missing values
ValDataUse<- read.csv("pml-testing.csv",na.strings=c("NA","#DIV/0!",""))
#step 2 - remove same cols ID earlier with > 50% NAs
ValDataUseClean <- ValDataUse[,-indColToRemove]
#step 3 - trim: do not need cols X (rowID) or user_name. They may effect the predictio
n model.
ValDataUseCleanTrim <- ValDataUseClean[,-c(1:2)]
#step 4 - remove the new_window column due to near zero variance
ValDataUseCleanTrimCOV<-subset(ValDataUseCleanTrim,select=-c(new_window))
#should have 20 rows and 57 columns
dim(ValDataUseCleanTrimCOV)
```

```
## [1] 20 57
```

Here are the predicted classe assignments on the validation dataset based on the random forest model:

```
#Ans for 2nd quiz
ValPred<-predict(FitRF,newdata=ValDataUseCleanTrimCOV)
ValPred
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```