Predicting Action from Wearable Device Data

Steeg Pierce

6/10/2020

Summary

We will be looking at data collected from wearable devices such as smart watches during bicep curls and looking to classify it into different degrees of proper form based on the levels given in the data. I have already loaded the data with a call to read.csv() for a training set and the quiz data. With the training set, we will create a machine learning algorithm that attemps to predict the "correctness" of the exercise.

Exploring the Data

By looking at the structure of the quiz dataframe, we can see that there are 160 variables and 100 of them are made up of majority NA. We can also see that of those with NA, the data they do carry are transformations of the other variables. As such, we can feel comfortable removing them from the variables with NA from the model. Additionally, the first 7 variables are related to time or user, which is not relevant to our predection. We will remove those ase well. We're left with 52 predictors.

Training the Model

The instructions from the assignment tell us that the variable classifying the "correctness" is 'classe', so we will be using that as the dependent variable. A principle component analysis will reduce the number of predictors while capturing most of the variation.

```r
set.seed(150)
inTrain <- createDataPartition(trainClean$classe, p = .6, list = FALSE)
training <- trainClean[inTrain, ]
temp <- trainClean[-inTrain, ]
set.seed(200)
inValidation <- createDataPartition(temp$classe, p = .5, list = FALSE)
validation <- temp[inValidation, ]
testing <- temp[-inValidation, ]
pre <- preProcess(training, method = 'pca', thresh = .9)
trainPCA <- predict(pre, newdata = training)
validPCA <- predict(pre, validation)
testPCA <- predict(pre, testing)
#Output number of principle components
pre$numComp
```

```
## [1] 18
```

We can see here that the number of predictors is drastically cut down via a principle component analysis. It went from 53 to 18. This will speed up our model.

Training the model will include *k-fold cross-validation* with 3 folds. More folds may help . Additionally, given the amount of data, too many folds may become too computationally demanding. We will fit three models to the training data.

First, we'll fit a *Linear Discriminant Analysis*. This will serve as a baseline.

```
set.seed(250)
tc <- trainControl(method = 'cv', number = 3)
fitLDA <- train(classe ~ ., method = 'lda', trControl = tc, data = trainPCA)
fitLDA
```

```
## Linear Discriminant Analysis
##
## 11776 samples
##    18 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 7851, 7851, 7850
## Resampling results:
##
##   Accuracy   Kappa
##   0.5015285  0.3648754
```

It says no pre-processing because we've already done that with a principle component analysis previously. We can see an in-sample accuracy estimate of approximately 50%. It's certainly better than random, but you wouldn't want to rely on a model like that.

Next up, we'll give the *Generalized Boosted Model* a go.

```
set.seed(300)
fitGBM <- train(classe ~ ., method = 'gbm', trControl = tc, data = trainPCA,
                verbose = FALSE)
fitGBM
```

```
## Stochastic Gradient Boosting
##
## 11776 samples
##    18 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 7851, 7850, 7851
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.5345620  0.4031459
##   1                  100      0.5856829  0.4715489
##   1                  150      0.6188861  0.5150485
##   2                   50      0.6390124  0.5400768
##   2                  100      0.7024462  0.6221144
##   2                  150      0.7387066  0.6686512
##   3                   50      0.6901325  0.6063839
##   3                  100      0.7567083  0.6915195
##   3                  150      0.7929686  0.7376357
##
## 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.
```

Accuracy is a bit better, but it's still not as good as we'd like. Still, we'll test it on the validation set to see how it does.

```r
predGBM <- predict(fitGBM, validPCA)
confusionMatrix(validation$classe, predGBM)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 974  43  31  64   4
##          B  78 571  69  23  18
##          C  53  55 538  24  14
##          D  30  24  70 498  21
##          E  25  47  53  30 566
##
## Overall Statistics
##
##                Accuracy : 0.8022
##                  95% CI : (0.7894, 0.8146)
##     No Information Rate : 0.2957
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7495
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8397   0.7716   0.7070   0.7793   0.9085
## Specificity            0.9486   0.9409   0.9538   0.9558   0.9530
## Pos Pred Value         0.8728   0.7523   0.7865   0.7745   0.7850
## Neg Pred Value         0.9337   0.9466   0.9312   0.9570   0.9822
## Prevalence             0.2957   0.1886   0.1940   0.1629   0.1588
## Detection Rate         0.2483   0.1456   0.1371   0.1269   0.1443
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.8941   0.8563   0.8304   0.8676   0.9308
```

Next up is the more powerful *Random Forest*.

```r
set.seed(350)
fitRF <- train(classe ~ ., method = 'rf', trControl = tc, data = trainPCA)
fitRF
```

```
## Random Forest
##
## 11776 samples
##    18 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 7850, 7851, 7851
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9485393  0.9348763
##   10    0.9446331  0.9299398
##   18    0.9386036  0.9223222
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

Accuracy for the random forest is quite high. We will test this one the validation set as well to get an idea of the out-of-sample error.

```r
predRF <- predict(fitRF, validPCA)
confusionMatrix(validation$classe, predRF)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1096   12    4    4    0
##          B   26  722   11    0    0
##          C    6   10  660    5    3
##          D    3    5   23  609    3
##          E    0   10    5    3  703
##
## Overall Statistics
##
##                Accuracy : 0.9661
##                  95% CI : (0.9599, 0.9715)
##     No Information Rate : 0.2883
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9571
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

```
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9691   0.9513   0.9388   0.9807   0.9915
## Specificity            0.9928   0.9883   0.9925   0.9897   0.9944
## Pos Pred Value         0.9821   0.9513   0.9649   0.9471   0.9750
## Neg Pred Value         0.9875   0.9883   0.9867   0.9963   0.9981
## Prevalence             0.2883   0.1935   0.1792   0.1583   0.1807
## Detection Rate         0.2794   0.1840   0.1682   0.1552   0.1792
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9809   0.9698   0.9657   0.9852   0.9930
```

Testing

Having seen that the *random forest* is the most accurate model on the validation set, we will use that on the testing set.

```
testRF <- predict(fitRF, testPCA)
confusionMatrix(testing$classe, testRF)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1095   12    4    3    2
##          B   20  726    9    0    4
##          C    5   10  655   14    0
##          D    4    2   28  606    3
##          E    0    6    6    4  705
##
## Overall Statistics
##
##                Accuracy : 0.9653
##                  95% CI : (0.9591, 0.9708)
##     No Information Rate : 0.2865
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9561
##
##  Mcnemar's Test P-Value : 0.0637
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9742   0.9603   0.9330   0.9665   0.9874
## Specificity            0.9925   0.9896   0.9910   0.9888   0.9950
## Pos Pred Value         0.9812   0.9565   0.9576   0.9425   0.9778
## Neg Pred Value         0.9897   0.9905   0.9855   0.9936   0.9972
## Prevalence             0.2865   0.1927   0.1789   0.1598   0.1820
## Detection Rate         0.2791   0.1851   0.1670   0.1545   0.1797
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9833   0.9749   0.9620   0.9776   0.9912
```

Having confirmed a high degree of accuracy for the model, we can use that to predict the quiz set.

```r
quizPCA <- predict(pre, testClean)
predict(fitRF, quizPCA)
```

```
##  [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
```