Executive Summary

The purpose of this analysis is to build a model which predicts the manner (the "classe") in which a barbell is being lifted, using data collected from accelerometers attached to the lifter's belt, forearm and arm.

The data was pre-processed before model fitting using a Principal Components Analaysis (PCA) approach, and the model was fit on training data using a random forest method. The model fit produced an out of sample error rate of less than 1%. This was cross-verified on a portion of the original source data which was excluded from model training and fitting.

The below sections show the steps in this analysis from data collection and processing, to final model fit and predictions on new data.

Load packages

Below code loads required r packages for this analysis.

```
library(caret)
library(rpart)
library(randomForest)
```

Data preparation and pre-processing

The data consists of measurements of speed and directions during lift repetitions, with over 150 variables being collected. Given that there will likely be significant correlations among some variables, we expect a Principal Components Analaysis (PCA) approach to be useful for pre-processing the data prior to model fitting. The purpose of PCA is to convert the variables into fewer principal components which are as uncorrelated as possible, but capture the majority of the observed data variance.

In the below code section, we first read the data from its source files and then trim to include only variables for which there are complete data that can be used to calculate variances and covariances - a prerequisite for PCA processing - and appear to have some bearing on the "classe" variable. The variables selected are based on a high level exploration of the data.

```
# read in the obvserved source data
data_all <- read.csv("./data/pml-training.csv", na.strings = c("#DIV/0!","NA",""), stringsAsFactors = F

# trim data to useful and complete measurement columns which can reliably support PCA preprocessing.
data_trim <- data_all[,c(7:11,37:49,60:68,84:86,102,113:124,140,151:160)]
data_trim$classe <- factor(data_trim$classe)
```

Data Preprocessing and Partitioning

As mentioned above, we will use a PCA approach to pre-process the data before model fitting. Before this step, we will first partition the data such that a random 75% of the observations are used to estimate a model fit. The remaining 25% will be used for cross-validating the model fit, by verifying how well predictions match with actual experience.

The PCA processing aims to translate the data into principal components which capture 80% of the original training data's variance. Based on this threshold, we are able to trim the variables down from 54 to 13 principal components.

```
## Split data in training and validation
set.seed(1234)
inTrain <- createDataPartition(data_trim$classe, p=0.75)[[1]]
training <- data_trim[inTrain,]
testing <- data_trim[-inTrain,]

# PCA preprocessing
set.seed(1234)
preProc <- preProcess(training[,-54], method="pca", thresh = 0.80)
train_PC <- predict(preProc, training)
head(train_PC)
```

```
##   classe      PC1      PC2      PC3       PC4      PC5       PC6
## 2      A 4.303660 1.743271 2.807234 0.9704351 1.317279 -1.854462
## 3      A 4.271426 1.759666 2.808159 0.9651489 1.250626 -1.790176
## 4      A 4.285464 1.753061 2.793391 0.9638470 1.263087 -1.828828
## 5      A 4.271597 1.811907 2.764332 0.9545165 1.276243 -1.872177
## 6      A 4.279944 1.782084 2.804942 0.9629650 1.259701 -1.810637
## 7      A 4.269117 1.732337 2.780548 0.9624812 1.252709 -1.802413
##          PC7      PC8        PC9      PC10      PC11        PC12      PC13
## 2 -0.2371163 2.827637 -0.1955286 0.4140572 -1.041765 -0.01947861 -3.298197
## 3 -0.2559084 2.853997 -0.1929336 0.4033038 -1.047775 -0.03270861 -3.322737
## 4 -0.2263089 2.824431 -0.2154065 0.4145439 -1.052166 -0.06627145 -3.304619
## 5 -0.2132730 2.831330 -0.1729816 0.4298634 -1.000438 -0.01283010 -3.326792
## 6 -0.2462728 2.859405 -0.1959444 0.3858806 -1.033581 -0.06825674 -3.329543
## 7 -0.2347551 2.869005 -0.1819164 0.4129094 -1.044556 -0.09307120 -3.361759
```

Model Fit

Given the non-linear, classification predictions being done, we have chosen to use a Random Forest model fit. The code below prints the results of this model fit, showing an out-of-sample error rate of about 3%.

```
model_Fit_rf <- randomForest(classe~., data = train_PC)
print(model_Fit_rf)
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = train_PC)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 3.27%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4102   26   31   17    9  0.01983274
## B   62 2721   53    7    5  0.04459270
## C    7   40 2479   38    3  0.03428126
## D   18   14   95 2278    7  0.05555556
## E    6   16   10   17 2657  0.01810791
```

Cross Validation

As a final check of the model's accuracy, we next apply it to the validation data which was carved out from the original data used to fit the model. This provides a more independent check of the model accuracy to the extent that data was not used for model fitting. Based on the Confusion Matrix output, the model predicts with about 96% accuracy.

```
test_PC <- predict(preProc, testing)
test_Predict_PC <- predict(model_Fit_rf, test_PC)
confusionMatrix(test_Predict_PC, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1365   23    3    1    1
##          B    6  907   20    6    3
##          C   10   12  813   33    5
##          D   11    5   16  763    8
##          E    3    2    3    1  884
##
## Overall Statistics
##
##                Accuracy : 0.9649
##                  95% CI : (0.9594, 0.9699)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9556
##  Mcnemar's Test P-Value : 5.223e-05
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9785   0.9557   0.9509   0.9490   0.9811
## Specificity            0.9920   0.9912   0.9852   0.9902   0.9978
## Pos Pred Value         0.9799   0.9628   0.9313   0.9502   0.9899
## Neg Pred Value         0.9915   0.9894   0.9896   0.9900   0.9958
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2783   0.1850   0.1658   0.1556   0.1803
## Detection Prevalence   0.2841   0.1921   0.1780   0.1637   0.1821
## Balanced Accuracy      0.9853   0.9734   0.9680   0.9696   0.9894
```