Practical Machine Learning Project

Summary

This paper explains my methodology for solving the final project for this course, Practical Machine Learning. First, the data sets are discussed, then some exploratory data analysis is discussed and its implications for the rest of the study. In support of this is a brief discussion of domain specific issues that impact the data choices, the algorithm and the underlying rationale. Third, algorithms are briefly discussed and than a specific general algorithm chosen, Support Vector Machines. This procedure is then applied with several different input parameter sets. Next, the specific code is presented amd execution issues addressed. Finally, the results are tabulated, compared and discussed.

Train and test data sets

Two important points: first, the data set called "pml-testing" is not the data set we will be doing machine learning testing with. It is actually the application data set that we apply the final model to for the quiz results. Second, the training data set is a large collection of time series, several measurements taken for each predictor for a span of time that performs the entire exercise. The test set is not like that.
The test set is a snapshot, one single observation of each predictor at the same time. The training set will be divided up into partitions by the cross-validation algorithm.

```
projectTrain.df <- read.csv("D:/coursera_localrepo/pml-training.csv")
projectTest.df <- read.csv("D:/coursera_localrepo/pml-testing.csv")
```

Exploratory data analysis

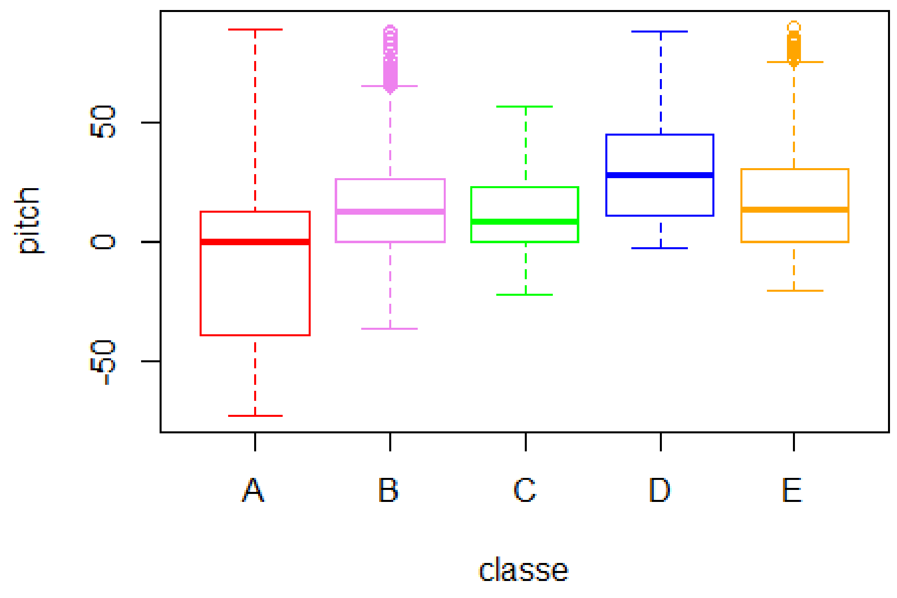
Consider two of the motions: lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) In these cases the y-axis measurement will stop short of the other three classes either going up or down. Hence these classes could be distinguised (possibly) from the others by analyzing y-axis predictors in detail (or in our case letting our machine learning algorithms do it for us). While not perfect this example combined with several physics observations gives the domain specific approach that we can aim for. The data streams in the training set will differ in important ways. Example: the maximum of the predictor $pitch_forearm is much smaller (*) for Class C (lifting dumbbell up only halfway) then the other classes as you would expect. Additionally, the min value is much higher (**) for Class D (lowering the dumbbell halfway) as you would expect.

```
          max $pitch_forearm            min $pitch_forearm        Class
            88                              -72                     A
            87                              -36                     B
            56    *                         -21                     C
            87                               -2   **                D
            89                              -20                     E
```

Also, based on the comment above about the test data being only single row snapshots, all columns/predictors that have NA in them for the test data set will be removed from both the training and test data set. Later on the first several columns with identifier data will be ignored when the algorithm is applied. At the end of this the training data set will be comprised of one predictor with 5 factor levels ("classe") and all other columns/predictors will be numeric.

```
    pitch_A <- projectTrain.df$pitch_forearm[projectTrain.df$classe=="A"]
    pitch_B <- projectTrain.df$pitch_forearm[projectTrain.df$classe=="B"]
    pitch_C <- projectTrain.df$pitch_forearm[projectTrain.df$classe=="C"]
    pitch_D <- projectTrain.df$pitch_forearm[projectTrain.df$classe=="D"]
    pitch_E <- projectTrain.df$pitch_forearm[projectTrain.df$classe=="E"]

    boxplot(pitch_A, pitch_B, pitch_C, pitch_D, pitch_E, ylab="pitch",
xlab="classe", names=c("A","B","C","D","E"), border=c("red", "violet",
"green", "blue", "orange"))
```



As you can imagine a data set with columns/predictors with characteristics like the above mini-table and plot seem ideal for an application that classifies by means of inserting a hyperplane between sets of data from different classes. Like a neural network or SVM.

```
    predRemove <- c(12:36,50:59,69:83, 87:101, 103:112, 125:139, 141:150 )
```

Domain Specific details - physics

The xyz axes are perpendicular and the data collected from the electronic sensors acts like linearly independent predictors for our purposes. Since linearly independent predictors are sort of an ideal: I will not be transforming the data streams into new variables or using principal components or the like.

Algorithm choice

After exploratory data analysis, I considered several algorithms/procedures for prediction. A guide that I use is from "The Elements of Statistical Learning" in section 10.7 "Off-the-shelf Procedures for Data Mining". The three procedures best at prediction are Neural Nets, SVMs and k-NN kernels.
All have similar problems handling mixed type data, however this data set has all numeric predictors, the only variable using factors (categorical data) is the output column "classe". With all of this in mind a multiple class SVM seems a logical choice, since SVMs require a categorical data set as the response or output. The package chosen is from the e1071 library.

Therefore, Support Vector Machines are my choice of an algorithm. I followed the examples in "An Introduction to Statistical Learning" on pages 363-367. Several R functions are also used: "scale" to get the data predictors in standarized form, and "tune" which performs a 10-fold cross-validation analysis of the data set with the "svm" function.

Coding and Execution

The actual code to run this example is short. It has a template in the above mentioned book. In the results section several other models and their results are listed for comparison. Their code is not listed here, just the final model which was then used to get the results for the quiz. The score on the quiz was 100%. The actual execution is moderate in CPU time. The 10-fold cross validation, the three cost levels and the 19000 point data set keep the computer running for several minutes.

```
## Warning: package 'e1071' was built under R version 3.1.3
```

```r
set.seed(1)
allDataSVM <- projectTrain.df[, c(-(1:6),-predRemove)]
testSVM <- projectTest.df[,c(-(1:6),-predRemove)]      # the test set of 20
rows

tune.out = tune(svm, classe~., data = allDataSVM, kernel = "polynomial",
degree = 3, ranges = list(cost = c(10, 100, 1000), scale = TRUE) )

summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost scale
##  1000  TRUE
```

```
##
## - best performance: 0.004128025
##
## - Detailed performance results:
##   cost scale       error  dispersion
## 1   10  TRUE 0.011823552 0.003167841
## 2  100  TRUE 0.004128103 0.002502224
## 3 1000  TRUE 0.004128025 0.002073130

bestmod = tune.out$best.model
yans <- predict(bestmod, testSVM)    # answers for the quiz
```

Discussion of the results

The results were instructive. Four models were run. The results of each are listed below with the final being the model that was chosen. First, the linear SVM model did not work well at all; the data just couldn't be seperated by a linear hyperplane. On the other hand, a quadratic model worked much better and a cubic model even better than that. Plus, using scaled data was a modest improvement over unscaled data. However, all of the first three models were used with a split of the projectTrain.df data set into two equal parts of 9811 points apiece. By using all of the data at once with a smaller portion used for testing - the CV or cross validation approach it was possible to reduce the model bias and improve the 4th and final model from the 1% error of the third model to .4% for the final model. That model was used in the prediction for the quiz. That is the only model that is listed in the coding section above.

From the results I believe that the out of sample error is about .4% for this data set. The chosen method does not seem to be out of line with the error estimates from simpler models (linear, quadratic, cubic) and the error estimates seemed to be converging in a sense.

Results

SVM linear

error was very large, linear model rejected

```
    ypred    A    B    C    D    E
        A 1154  497  212  316  474
        B  564 1107  481  237  351
        C  501  112  707  128  341
        D  286   97  179  783  171
        E  305   69  170  141  428
```

SVM polynomial degree = 2

error = 2.5% (246/9811 wrong) (no scaling)

```
    ypred    A    B    C    D    E
        A 2778   15    0    6   12
        B    21 1822    32     2    26
        C     0    11 1734    36    16
        D     3     0    23 1514    16
        E     2     6     4    15 1717
```

SVM polynomial degree = 3

error = 1% (109/9811 wrong) (with scaling)

```
    ypred     A     B     C     D     E
        A 2797    16     0     0     2
        B     6 1832     5     0    11
        C     1     6 1771    16    18
        D     0     0    15 1554     8
        E     0     0     2     3 1748
```

SVM polynomial degree = 3 and 10-fold cross validation

error = .4% (with scaling)

```
  Parameter tuning of 'svm':

    - sampling method: 10-fold cross validation

  - best parameters:
    cost
  1000

  - best performance: 0.004128025

  - Detailed performance results:
    cost       error  dispersion
  1   10 0.011823552 0.003167841
  2  100 0.004128103 0.002502224
  3 1000 0.004128025 0.002073130
```