Practical Machine Learning - Prediction Assignment Write Up

Summary of Problem

We are provided two datasets, *training data* and *testing data*. The output we are trying to predict is 'classe'. We produce a prediction model that runs against the testing data and classifies the output.

The output 'classe' has five categories. A or 1 indicates exercise done correctly, and B,C,D,E (2,3,4,5) exercise not done correctly. Our task is determine the "exercise done correctly" categorisation.

Solution

In order to predict against the testing dataset we need to understand what inputs are provided in the testing dataset. It is no good using inputs that are not available in the testing dataset, even though they may be available in the training dataset.

There are some obvious columns to immediately drop in the testing and training datasets such as *'X','user_name', 'raw_timestamp_part_1', 'raw_timestamp_part_2', 'cvtd_timestamp', 'new_window', 'num_window'*, ans *'problem_id'*.

My approach was then to find all columns in the testing dataset that contained numeric values. At that point we have a set of indicators that we know can be used for testing.

Next we review the training data. We again extract only numeric columns and also drop rows where *'new_window=yes'* - this because in testing data *'new_window=no'*. Next we extract common indicators between testing and training datasets. Having done that we know that our classification model will use indicators that are available in the testing dataset.

The training data is in a set sequence and because of that we first randomize the row order.

We now split the training data into two datasets using createDataPartition. A "genuine" training set (70%) and a validation set (30%). Later we use the validation set to test how well we believe the prediction model performs.

Next we fit the training data using a random forest tree model. In practice the caret *train()* function just took far too long and I abandoned that approach. Using *randomForest()* function directly was computationally much quicker.

Model Performance

The *modelFit*, the output of *randomForest(),* for the random forest model is shown below:

```
        OOB estimate of  error rate: 0.52%
Confusion matrix:
     A    B    C    D    E  class.error
A 3830    2    0    0    0 0.0005219207
B   13 2544    6    0    0 0.0074131877
C    0   17 2350    1    0 0.0076013514
D    0    0   20 2200    3 0.0103463788
E    0    0    2    6 2458 0.0032441200
```

As can be seen the classification appears to be extremely good.

Now we test model performance against the validation set. We compare prediction with actual. Before we do that, the ‘classe’ factor variable is adjusted from A (1), B (2), C (3), D (4), E (5) to A (1), (B, C, D, E) (2). Remember we are testing whether the exercise is done correctly as an A (1). We bundle “not done correctly” into a single factor (2).

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2
         1 1634    6
         2    5 4119

               Accuracy : 0.9981
                 95% CI : (0.9966, 0.999)
    No Information Rate : 0.7156
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9953
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9969
            Specificity : 0.9985
         Pos Pred Value : 0.9963
         Neg Pred Value : 0.9988
             Prevalence : 0.2844
         Detection Rate : 0.2835
   Detection Prevalence : 0.2845
      Balanced Accuracy : 0.9977

       'Positive' Class : 1
```

This is further summarised in the confusion matrix below from the validation dataset.

Confusion Matrix

	Actual A	**Actual B,C,D,C**
Predicted A	True Positive 1634	False Positive 6
Predicted B,C,D,C	False Negative 5	True Negative 4119

From the validation set the likelihood of an incorrect classification is around 0.2% - *(False Positive + False Negative)/ Total*. That out-of-sample error rate seems almost too good to be true, but with some certainty we can state the error rate is better than 0.5%.

Prediction On 20 Test Cases

The usefulness of a model is in how well it predicts against the testing data. All 20 cases were predicted correctly. We can conclude that the model is working satisfactorily.

Implementation Pain

While the above summary may imply everything was smooth sailing, this was not the case.

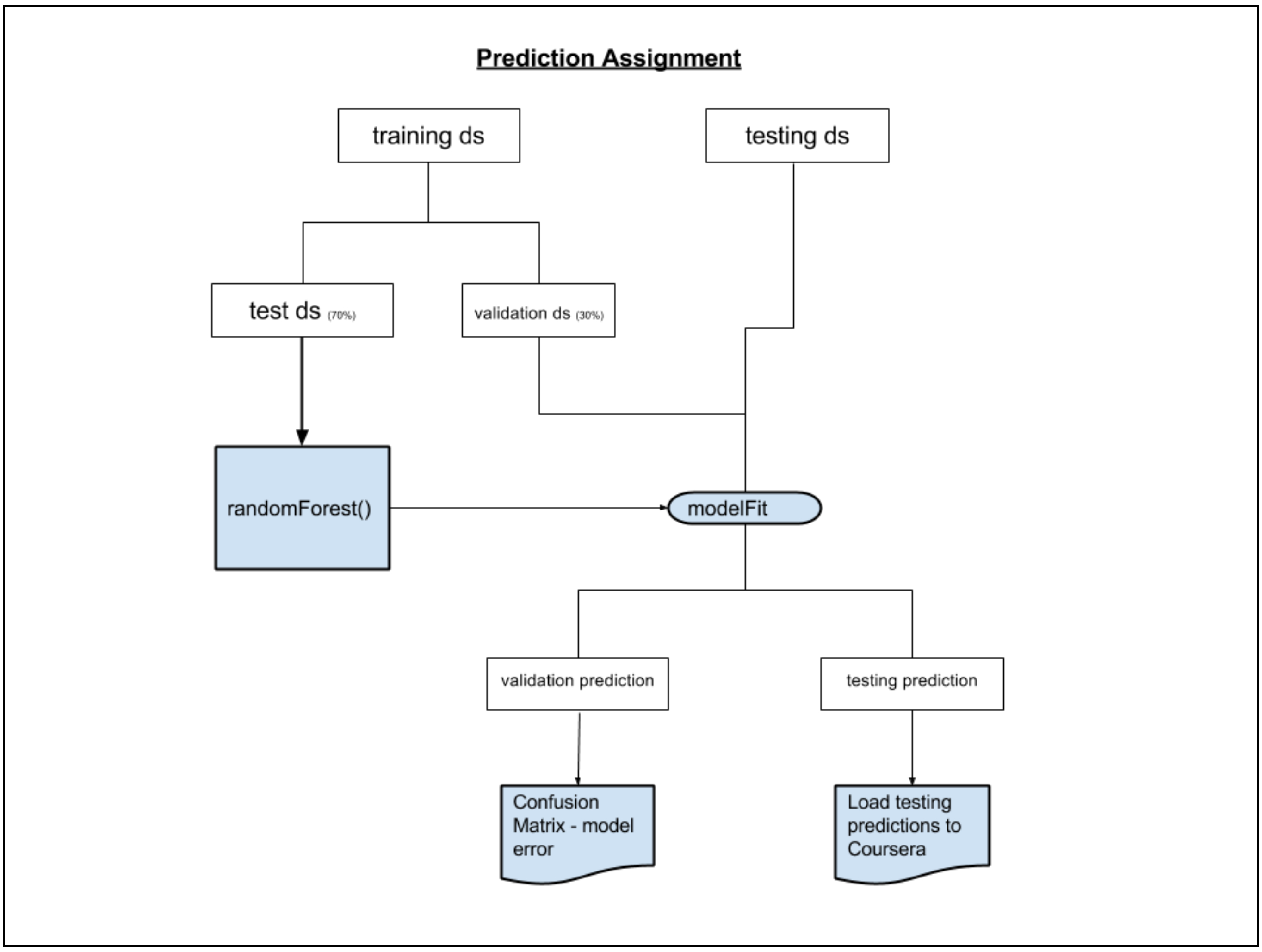
Initially I proceeded with training data pre-processing without looking at the testing data. When I needed to predict using the test data, everything fell apart.

I then realised I had to first investigate how the test data looked, and use only a subset of those inputs in the training data.

I also realised to validate the model, and get a more reliable prediction for the out-of-sample error rates, the testing data needed to be split into *testing* and *validation* datasets.

The performance of the caret *train()* function against my training data was very poor, and that approach had to be abandoned and replaced with the *randomForest()* function. It took a few nights to determine that.

Diagram of Prediction Assignment



Working R Code

Code is provided in project02.R, which does the bulk of the heavy lifting. Output of files for Coursera is in answers.R.

Indicator (Input) Column Names Used In the Model

```
 [1] "roll_belt"            "pitch_belt"           "yaw_belt"
 [4] "total_accel_belt"     "gyros_belt_x"         "gyros_belt_y"
 [7] "gyros_belt_z"         "accel_belt_x"         "accel_belt_y"
[10] "accel_belt_z"         "magnet_belt_x"        "magnet_belt_y"
[13] "magnet_belt_z"        "roll_arm"             "pitch_arm"
[16] "yaw_arm"              "total_accel_arm"      "gyros_arm_x"
[19] "gyros_arm_y"          "gyros_arm_z"          "accel_arm_x"
[22] "accel_arm_y"          "accel_arm_z"          "magnet_arm_x"
[25] "magnet_arm_y"         "magnet_arm_z"         "roll_dumbbell"
[28] "pitch_dumbbell"       "yaw_dumbbell"         "total_accel_dumbbell"
[31] "gyros_dumbbell_x"     "gyros_dumbbell_y"     "gyros_dumbbell_z"
[34] "accel_dumbbell_x"     "accel_dumbbell_y"     "accel_dumbbell_z"
[37] "magnet_dumbbell_x"    "magnet_dumbbell_y"    "magnet_dumbbell_z"
[40] "roll_forearm"         "pitch_forearm"        "yaw_forearm"
[43] "total_accel_forearm"  "gyros_forearm_x"      "gyros_forearm_y"
[46] "gyros_forearm_z"      "accel_forearm_x"      "accel_forearm_y"
[49] "accel_forearm_z"      "magnet_forearm_x"     "magnet_forearm_y"
[52] "magnet_forearm_z"     "classe"
```

Note: *‘classe’* is an output