Practical Machine Learning - Course Project

Khalil H Najafi

01/07/2019

Introduction & Setup

Per the outline of the course project, this write up will walk through the full machine learning workflow for a classification model on the letter grade of various exercises.

Already loaded in the background were the following packages: `data.table`, `tidyverse`, `caret`, and `randomForest`.

Dataset

Raw Dataset Import

Per the outline, we load the datasets to be used from the source.

```
## Training Dataset
data_training <- fread("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv")
data_training <- as.data.frame(data_training)

## Testing Dataset
data_testing <- fread("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv")
data_testing <- as.data.frame(data_testing)
```

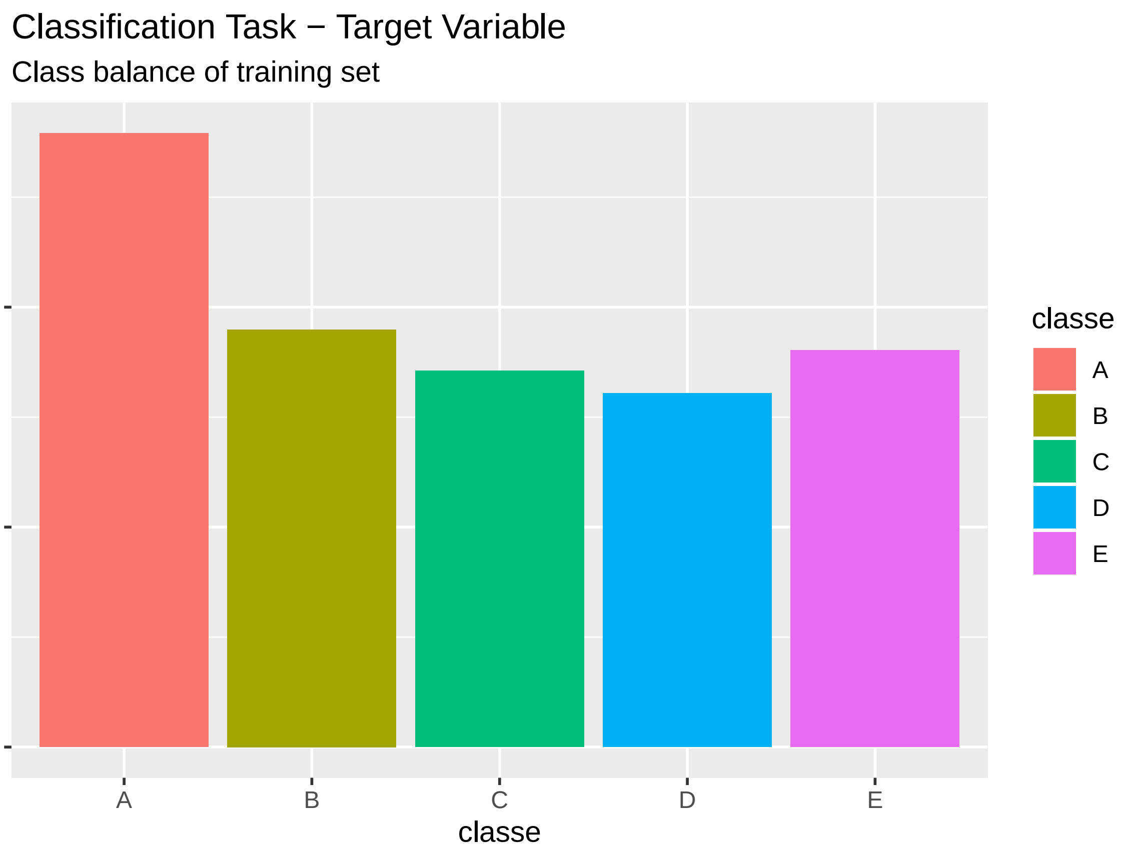
Dataset Tidying & Preprocessing

The framework applied to the tidying and preprocessing portion of the model development can be summarized as follows:

- Reproducibility - use `set.seed` function to ensure reproducibility
- Review Variables - manually review to remove any unwanted/unhelpful identifier variables and check for missing values (`NA`), zero variance, and `NaN`s
- Dependent Variable - review and process the outcome variable
- Basic Feature Selection - review the dataset to check for any issues in the variables
- Preprocessing - (if applicable) perform preprocessing based on the exploration
- Cross Validation & Splitting - (if applicable) split the training dataset for model building and optimization

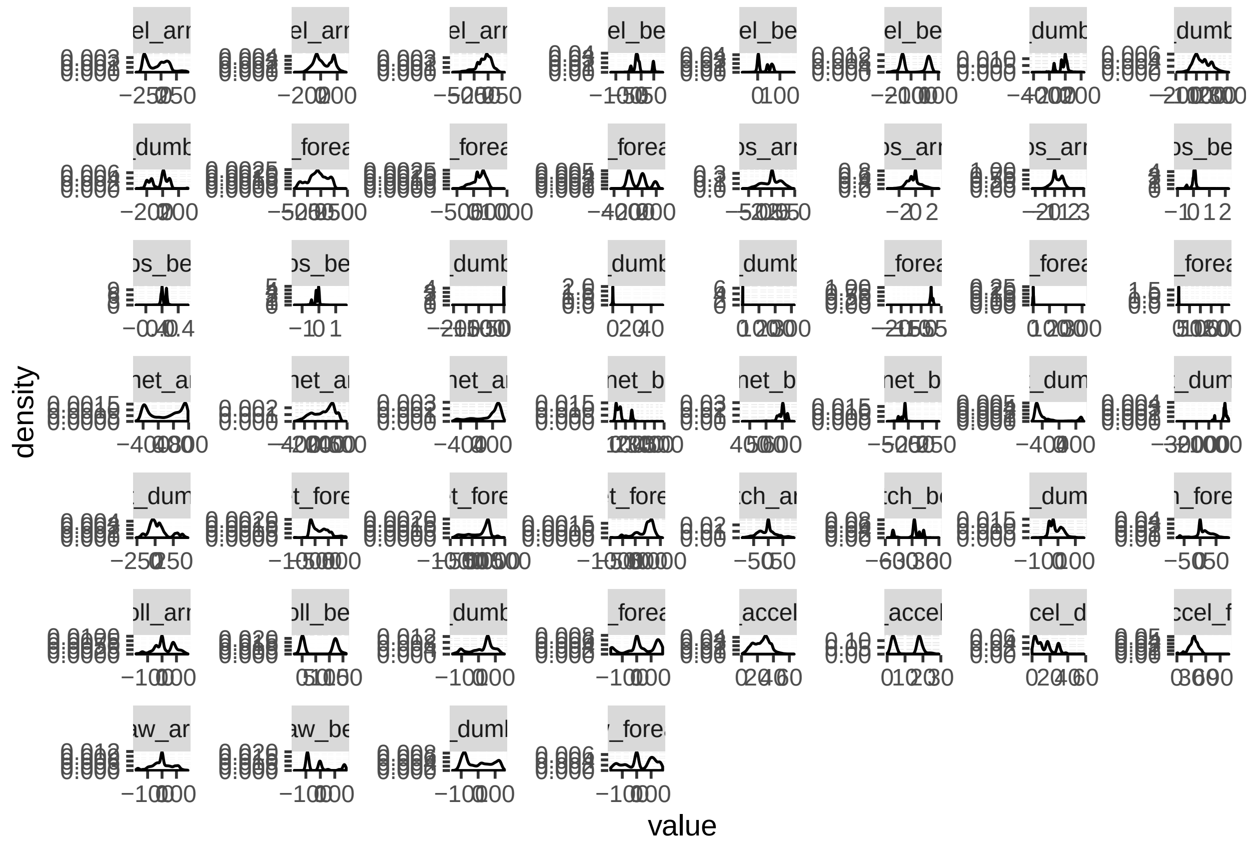
```
## Reproducibility (set seed)
set.seed(2604)

## Review Variables
# Remove identification and timestamp variables as they offer no predictive use
id_vars <- names(data_training)[1:7]
data_training <- data_training %>%
        select(-id_vars)

# Check and remove variables with no data as they cannot assist and potentially harm the model's effect
```

```r
check_NA <- function(x) {sum(is.na(x))}

data_training_NAs <- apply(data_training, 2, check_NA)
data_training <- data_training[,which(data_training_NAs == 0)]

## Dependent Variable
# Classification Task
# Given the objective of classification, we'll conver the variable type to a factor
data_training <- data_training %>%
        mutate(classe = as.factor(classe))

# Plot the outcome variable to check class balance, missing values, etc
ggplot(data_training, aes(classe)) +
        geom_bar(aes(fill = classe)) +
        labs(title = "Classification Task - Target Variable",
             subtitle = "Class balance of training set",
             y = "")
```



```r
## Basic Feature Selection
# We can explore the distribution of the independent variables to determine which will be useful in our
feature_data <- data_training %>%
        select(-classe) %>%
        gather() %>%
        filter(!is.na(value))
```

```
ggplot(feature_data, aes(value)) +
        geom_density(alpha = 2/3) +
        facet_wrap(~ key, scales = "free")
```



```
## Cross Validation and Splitting
# Given the size of the training dataset, we can easily split this further into a model building subset
index_training <- createDataPartition(y = data_training$classe,
                                      p = 0.7,
                                      list = F, )

data_training_build <- data_training %>%
        slice(index_training)
data_training_validation <- data_training %>%
        slice(-index_training)
```

Model Building & Optimization

With the dataset reviewed and processed, we can now build our classification model and optimize it before applying it to the test set.

Given the exploration of the dataset above, and especially as there is no special domain knowledge within training and fitness data, we will employ as many of the variables as possible. Thinking further from a product company's perspective (such as the companies that build these devices) they would rank accuracy higher than interpretability, and efficiency may range based on whether the models are built on the device, are regularly updated via cloud services, etc. From my own knowledge I know there is an increasing tolerance for computational resources both from the growing power of the cloud and faster Internet bandwidth. With all of the above we'll go with a random forest model as it trades accuracy for interpretability and is moderately expensive in computation. As a caveat, since we're building the model locally we'll limit the number of models, repeated cross validations, and repeated samples in the random forest model and check the statistics before increasing these parameters:

Model Build

```
## Simpler single random forst model fit
rf_model <- randomForest(classe ~ .,
                         data = data_training_build)

rf_model
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = data_training_build)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.43%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3905    1    0    0    0 0.0002560164
## B    9 2644    5    0    0 0.0052671181
## C    0   13 2382    1    0 0.0058430718
## D    0    0   22 2229    1 0.0102131439
## E    0    0    1    6 2518 0.0027722772
```

Model Evaluation & Optimization

Given the very low out of bounds estimate, moderate tree count, and decent computation time, we can apply the model to our validation set:

```
predictions_validation <- predict(rf_model, data_training_validation)
confusionMatrix(predictions_validation, data_training_validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    5    0    0    0
##          B    0 1132    5    0    0
##          C    0    2 1019   10    0
##          D    0    0    2  951    2
##          E    1    0    0    3 1080
##
## Overall Statistics
##
##                Accuracy : 0.9949
##                  95% CI : (0.9927, 0.9966)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9936
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9994   0.9939   0.9932   0.9865   0.9982
## Specificity            0.9988   0.9989   0.9975   0.9992   0.9992
## Pos Pred Value         0.9970   0.9956   0.9884   0.9958   0.9963
## Neg Pred Value         0.9998   0.9985   0.9986   0.9974   0.9996
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2843   0.1924   0.1732   0.1616   0.1835
## Detection Prevalence   0.2851   0.1932   0.1752   0.1623   0.1842
## Balanced Accuracy      0.9991   0.9964   0.9954   0.9929   0.9987
```

Accuracy on the validation set is sufficiently high at 0.9949023, and from the table of observed versus predicted we see that most errors occurred in the D-grade classification which correlates with the grade that had the lowest number of observations in the training set (fewest cases).

We can take this as a fair estimate of the out of sample error and proceed with classifying on the testing set without any further optimization of the model:

Model Application

Here are the predictions of the grades for the test set activities using our random forest classification model:

```
predictions <- predict(rf_model, data_testing)
predictions
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```

Conclusion

Our classification model is based on a random forest built from the training data using 53 variables where there were no missing observations or zero variance. We split the training data 70/30 for building and validation respectively and saw an acceptable accuracy on our validation set to give us confidence with the model's capability on out of sample test data.

Some bonus point considerations:

- Are there variables with more explanatory power that we are missing from the dataset?
- How extensive are the samples of poorly executed activities considering the userbase (population) of our product? In other words, are the ways we modelled A- through D-grade activities the common ways that our users perform these activities?

    - As an example, if these modelled activities were done by an average group but our product is mostly used by professional athletes

- Do our users want to understand more about the interactions and nuances of well executed activities (ie, is our assumptions of accuracy over interpretability valid)

All feedback and comments are welcome and are in fact desired.