Practical Machine Learning - Course Project

Dinesh Ghimire

May 13, 2016

Data Sources The training data for this project are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The test data are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

Introduction

For this project, we are given data from accelerometers on the belt, forearm, arm, and dumbell of 6 research study participants. Our training data consists of accelerometer data and a label identifying the quality of the activity the participant was doing. Our testing data consists of accelerometer data without the identifying label. Our goal is to predict the labels for the test set observations.

Below is the code I used when creating the model, estimating the out-of-sample error, and making predictions. I also include a description of each step of the process.

Data Preparation I load the caret package, and read in the training and testing data: `{r, echo=TRUE} #install.packages('caret', repos='http://cran.rstudio.com/') library(caret) library(lattice) library(ggplot2)` ## Loading required package: lattice ## Loading required package: ggplot2 `{r, echo=TRUE} ptrain <- read.csv("D:/MachineLearning/pml-training.csv") ptest <- read.csv("D:/MachineLearning/pml-testing.csv")` Because I want to be able to estimate the out-of-sample error, I randomly split the full training data (ptrain) into a smaller training set (ptrain1) and a validation set (ptrain2):

`{r, echo=TRUE} set.seed(10) inTrain <- createDataPartition(y=ptrain$classe, p=0.7, list=F) ptrain1 <- ptrain[inTrain, ] ptrain2 <- ptrain[-inTrain, ]` I am now going to reduce the number of features by removing variables with nearly zero variance, variables that are almost always NA, and variables that don't make intuitive sense for prediction. Note that I decide which ones to remove by analyzing ptrain1, and perform the identical removals on ptrain2:

"'{r, echo=TRUE} # remove variables with nearly zero variance nzv <- nearZeroVar(ptrain1) ptrain1 <- ptrain1[, -nzv] ptrain2 <- ptrain2[, -nzv]

remove variables that are almost always NA

mostlyNA <- sapply(ptrain1, function(x) mean(is.na(x))) > 0.95 ptrain1 <- ptrain1[, mostlyNA==F] ptrain2 <- ptrain2[, mostlyNA==F]

remove variables that don't make intuitive sense for prediction (X, user_name, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timestamp), which happen to be the first five variables

ptrain1 <- ptrain1[, -(1:5)] ptrain2 <- ptrain2[, -(1:5)] `#Model Building I decided to start with a Random Forest model, to see if it would have acceptable performance.  I fit the model` `on ptrain1, and instruct the "train" function to use 3-fold cross-validation to select optimal tuning parameters for the model.`{r, echo=TRUE} # instruct train to use 3-fold CV to select optimal tuning parameters fitControl <- trainControl(method="cv", number=3, verboseIter=F)

fit model on ptrain1

fit <- train(classe ~ ., data=ptrain1, method="rf", trControl=fitControl) "'

"'{r, echo=TRUE} library(randomForest) ## Loading required package: randomForest ## randomForest 4.6-10 ## Type rfNews() to see new features/changes/bug fixes. # print final model to see tuning parameters it chose

fit$finalModel "'

I see that it decided to use 500 trees and try 27 variables at each split.

Model Evaluation and Selection

Now, I use the fitted model to predict the label ("classe") in ptrain2, and show the confusion matrix to compare the predicted versus the actual labels: "'{r, echo=TRUE} # use model to predict classe in validation set (ptrain2) preds <- predict(fit, newdata=ptrain2)

show confusion matrix to get estimate of out-of-sample error

confusionMatrix(ptrain2$classe, preds)

"'

The accuracy is 99.8%, thus my predicted accuracy for the out-of-sample error is 0.2%.

This is an excellent result, so rather than trying additional algorithms, I will use Random Forests to predict on the test set.

Re-training the Selected Model

Before predicting on the test set, it is important to train the model on the full training set (ptrain), rather than using a model trained on a reduced training set (ptrain1), in order to produce the most accurate predictions. Therefore, I now repeat everything I did above on ptrain and ptest:

"'{r, echo=TRUE} # remove variables with nearly zero variance nzv <- nearZeroVar(ptrain) ptrain <- ptrain[, -nzv] ptest <- ptest[, -nzv]

remove variables that are almost always NA

mostlyNA <- sapply(ptrain, function(x) mean(is.na(x))) > 0.95 ptrain <- ptrain[, mostlyNA==F] ptest <- ptest[, mostlyNA==F]

remove variables that don’t make intuitive sense for prediction (X, user__name, raw__timestamp__part__1, raw__timestamp__part__2, cvtd__timestamp), which happen to be the first five variables

ptrain <- ptrain[, -(1:5)] ptest <- ptest[, -(1:5)]

re-fit model using full training set (ptrain)

fitControl <- trainControl(method="cv", number=3, verboseIter=F) fit <- train(classe ~ ., data=ptrain, method="rf", trControl=fitControl) "'

Making Test Set Predictions

Now, I use the model fit on ptrain to predict the label for the observations in ptest, and write those predictions to individual files: "'{r, echo=TRUE} # predict on test set preds <- predict(fit, newdata=ptest)

convert predictions to character vector

preds <- as.character(preds)

create function to write predictions to files

pml_write_files <- function(x) { n <- length(x) for(i in 1:n) { filename <- paste0("problem_id_", i, ".txt") write.table(x[i], file=filename, quote=F, row.names=F, col.names=F) } }

create prediction files to submit

pml_write_files(preds) #