Pracctical Machine Learning: Project Write-up

1. Overview

This document is for the writeup part of the course project of the Practical Machine Learning course, and it presents a machine learning model trained for predicting personal activity of 6 participants using information from accelerometers on the belt, forearm, arm, and dumbell.

2. Data preprocessing

After loading the data, check the columns first. The first column (1,2, . . .) is about row numbers and obviously not a feature. And the columns that are mostly empty or NA cells are also not proper features due to the difficulty or infeasibility of estimating the cell values, like those starting with "kurtosis_", "skewness_", "max_", "min_", . . . etc. These columns are manually examined and then also removed.

```
library("caret")
```

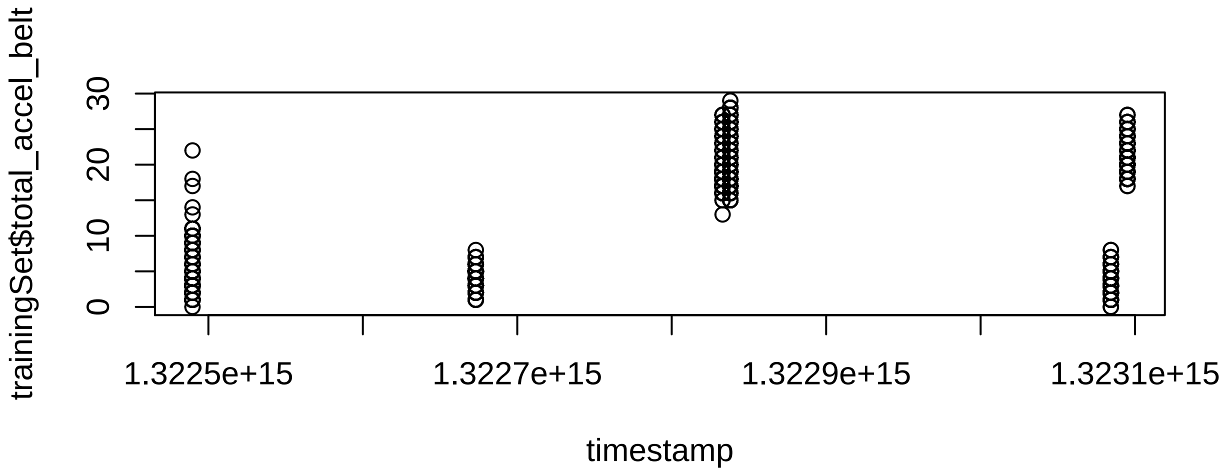
```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
trainingSet = read.csv("pml-training.csv");
testSet = read.csv("pml-testing.csv");
trainingSet$X = testSet$X = NULL;

colsToRemove = grep(names(trainingSet), pattern = "^(kurtosis|skewness|max|min|amplitude|var|arg|stddev
trainingSet = trainingSet[, -colsToRemove];
testSet = testSet[, -colsToRemove];
```

Subsequently, examine the remaining rows. Due to lack of online documentation, the meaning of the time related columns are not clear. It can be observed and guessed that raw_timestamp_part_1 and raw_timestamp_part_2 should be combined together to a single timestamp info.

```
#plot(trainingSet$raw_timestamp_part_2, trainingSet$roll_arm)
cvtdtime = as.POSIXct(strptime(trainingSet$cvtd_timestamp, "%d/%m/%Y %H:%M"))
#plot(cvtdtime, trainingSet$classe);
#plot(cvtdtime, trainingSet$pitch_belt);
cvtdtime= as.POSIXct(strptime(substr(trainingSet$cvtd_timestamp, 12, 16) , "%H:%M"))
#plot(cvtdtime, trainingSet$classe);
#max(trainingSet$raw_timestamp_part_2) = 998801;
timestamp = trainingSet$raw_timestamp_part_1 * 1000000 + trainingSet$raw_timestamp_part_2;
#plot(timestamp, trainingSet$classe)
plot(timestamp, trainingSet$total_accel_belt)
```



After quickly making some plots about relationships between time and classe and a few other variables, some pattern, as illustrated below, is found between timestamp and the 'total_accel_belt' variable. This shows that timestemp may reveal some non-random info and can be taken as a feature. And no patterns are found between cvtd_timestamp and classe or other variables, so cvtd_timestamp is also removed.

In addition, categorical variables are converted to variables with values 0 or 1. And the values of "new_window" in the testSet are all "no", plus the proportion of "yes" rows is small, so rows with "yes" are removed, and then the 'new_window' columns of the trainingSet and testSet can be removed. Finally, the training and testing sets are preprocessed like the following snippet. Moreover, it's not sure whether user_name is a proper feature or not, and it's experimented during cross validation by including and excluding user_name and check their kappa.

```r
trainingSet = subset(trainingSet, new_window == "no");
trainingSet$new_window = testSet$new_window = NULL

timestamp = trainingSet$raw_timestamp_part_1 * 1000000 + trainingSet$raw_timestamp_part_2;
names(trainingSet)[2] = "timestamp"; trainingSet$timestamp = timestamp;
trainingSet$raw_timestamp_part_2 = trainingSet$cvtd_timestamp = NULL;
dmvs = dummyVars("~ .", data = trainingSet[, - which(names(trainingSet) == "classe")]);
classe =trainingSet$classe;
trainingSet = data.frame(predict(dmvs, newdata = trainingSet), classe);


timestamp = testSet$raw_timestamp_part_1 * 1000000 + testSet$raw_timestamp_part_2;
names(testSet)[2] = "timestamp"; testSet$timestamp = timestamp;
testSet$raw_timestamp_part_2 = testSet$cvtd_timestamp = NULL;
dmvs = dummyVars("~ .", data = testSet[, - which(names(testSet) == "problem_id")]);
problem_id = testSet$problem_id;
testSet = data.frame(predict(dmvs, newdata = testSet), problem_id);
```

3. How the model is built

The caret package is a useful wrapper that wraps many machine learning packages. Many commonly used classfification methods are provided, such 'svmLinear', 'lda'(linear discriminant analysis), 'rf', 'knn'. Thus, I use caret package to train model for this project. These four methods are tried, and the optimal model based on kappa statistic evaluated using cross validation will be taken as the final model. With multiple repeats, the variation of accuracies due to sampling difference will be small.

```
tc = trainControl(method = "repeatedcv", number = 5, repeats = 3,
                  savePred=T, classProb=T, preProcOptions = list(thresh = 0.95))

meth0ds = c("svmLinear", "lda", "rf", "knn");
models = list();

for (i in 1:4){
  models[[i*2-1]] <- train(classe~., data = trainingSet, method = meth0ds[i], preProcess = "pca", trCon
  models[[i*2-1]]$results[c("Accuracy", "Kappa")]
  models[[i*2]] <- train(classe~., data = trainingSet[, -c(1:6)], method = meth0ds[i], preProcess = "pca
  models[[i*2]]$results[c("Accuracy", "Kappa")]
}
```

```
## line search fails -1.216547 0.08870835 1.07803e-05 -1.058251e-06 -9.537503e-09 -1.028706e-09 -1.0172

## Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels
## = param): kernlab class prediction calculations failed; returning NAs

## Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =
## param): kernlab class probability calculations failed; returning NAs

## Warning in data.frame(..., check.names = FALSE): row names were found from
## a short variable and have been discarded

## line search fails -1.363602 -0.0748573 2.942411e-05 1.505686e-06 -3.369495e-08 -1.525661e-08 -1.0144

## Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels
## = param): kernlab class prediction calculations failed; returning NAs

## Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =
## param): kernlab class probability calculations failed; returning NAs

## Warning in data.frame(..., check.names = FALSE): row names were found from
## a short variable and have been discarded

## line search fails -1.228126 0.0692077 2.53674e-05 -1.729625e-06 -2.274913e-08 -3.614059e-09 -5.70835

## Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels
## = param): kernlab class prediction calculations failed; returning NAs

## Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =
## param): kernlab class probability calculations failed; returning NAs

## Warning in data.frame(..., check.names = FALSE): row names were found from
## a short variable and have been discarded

## line search fails -1.359611 -0.07336952 1.54355e-05 1.112342e-06 -1.73465e-08 -8.096127e-09 -2.76757
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```r
for (i in 1:8){
  print(models[[i]]$results[c("Accuracy", "Kappa")])
}
```

```
##    Accuracy     Kappa
## 1 0.5104464 0.3688201
##    Accuracy     Kappa
## 1 0.5037877 0.3611504
##    Accuracy    Kappa
## 1 0.5285347 0.403118
##    Accuracy     Kappa
## 1 0.5277009 0.4018246
##    Accuracy     Kappa
## 1 0.9815783 0.9766936
## 2 0.9739280 0.9670159
## 3 0.9743271 0.9675199
##    Accuracy     Kappa
## 1 0.9815429 0.9766482
## 2 0.9716726 0.9641645
## 3 0.9722624 0.9649107
##    Accuracy     Kappa
## 1 0.9656012 0.9564798
## 2 0.9548635 0.9428921
## 3 0.9442647 0.9294765
##    Accuracy     Kappa
## 1 0.9663647 0.9574441
## 2 0.9557137 0.9439650
## 3 0.9452191 0.9306813
```

Results show that the random forest model models[[5]]'s performance is nearly perfect, plus it uses more features. Thus, models[[5]] is taken as the final model.

3.1 How cross validation is used

In the 'train' function of caret, the user can configure cross validation with the 'trainControl' parameter. The following is a train control for 5-fold CV with repeated sampling 3 times.

```
tc = trainControl(method = "repeatedcv", number = 5, repeats = 3,
                  savePred=T, classProb=T, preProcOptions = list(thresh = 0.95))
```

3.2 What the expected out of sample error is

The expected out of sample error can be estimated with the following snippet.

```
inTrain = createDataPartition(trainingSet$classe, p=.8, list=F)
trainPart = trainingSet[inTrain,]
testPart = trainingSet[-inTrain,]

model = train(classe~., data = trainPart, method = "rf", preProcess = "pca", trControl = tc);
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```r
predictions <- predict(object = model, newdata = testPart);
cm = confusionMatrix(data = predictions, reference = testPart$classe);
print(paste("error =", 1 - as.numeric(cm$overall[1])));
```

```
## [1] "error = 0.0156209320489455"
```

The result shows that expected out of sample error would be around 1% to 2%.

3.3 Why the choices are made

The choice of experimenting 4 different classification methods is for finding a better model. The size of the training set is not huge, and my computation resources and time allows me to made this choice.

The choice of using pca to preprocess features is that pca is an empirically very effective method for dimension reduction, which usually improves the performance of the trained model.

Test result on the test set

```r
predictions = predict(object = models[[5]], newdata = testSet);
predictions
```

```
##  [1] B A C A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```