Exercising monitoring

Jorge Monforte González

16 de julio de 2016

Executive summary

The objetive of this paper is fit a machine learning algorithm that is able to predict the *classe* (stored in column 160 of the data) of the exercise given the sensors values.

Dependencies

We will be using these libraries during or model fitting.

```
library(caret)
library(neuralnet)
```

Data preparation

First we load the data from the files in two data.frames, and remove the defective and metadata that is not useful for the training.

```
training.base <- read.csv('pml-training.csv')
testing.base <- read.csv('pml-testing.csv')

# Remove spurious columns

metadata_columns <- 1:7
div_0_columns <- get_div0_columns(training.base)
testing_na_columns <- get_na_columns(testing.base)
problem_id_column <- 160

training.clean <- training.base[,-1*c(metadata_columns, div_0_columns, testing_na_columns)]
testing.clean <- testing.base[,-1*c(metadata_columns, div_0_columns, testing_na_columns, problem_id_colu
```

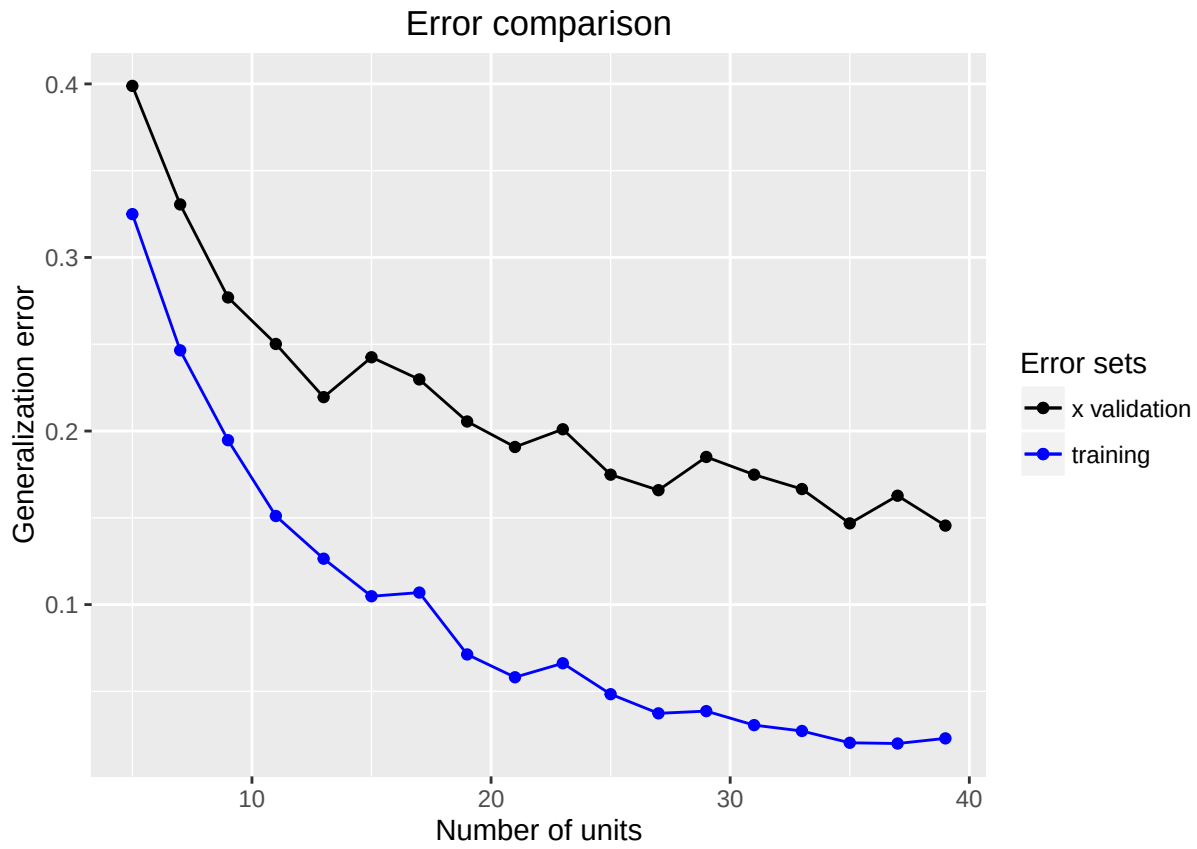
Network design and cross validation strategy

For designing the neural network we have to evaluate the size of the hidden layer so it minimizes the cross validation error. For doing this we take a subsample of the data and train multiple geometries and then we search for the neural network geometry that minimizes the error.

```
set.seed(543210)
# Subsample training values for faster training work
sampled_percentage <- 0.2
sampled <- sample(1:nrow(training.clean), round(sampled_percentage*nrow(training.clean)))
subsampled <- training.clean[sampled,]

# Impute missing values
```

```
impute.fit <- preProcess(subsampled, method=c("knnImpute"),
thresh=0.95)
subsampled <- predict(impute.fit, subsampled)

# Partition data
inTrain <- createDataPartition(y=subsampled$classe, p=0.60, list=FALSE)
training <- subsampled[inTrain,]
xvalidation <- subsampled[-inTrain,]

# Split into target features sets
classe_column <- which(colnames(training) == 'classe')
training.Y <- training[,classe_column]
xvalidation.Y <- xvalidation[,classe_column]
training.features <- training[,-classe_column]
xvalidation.features <- xvalidation[,-classe_column]

# Reduce the number of features using PCA
pre.fit <- preProcess(training.features, method=c("pca"),
thresh=0.99)
training.X <- predict(pre.fit, training.features)
xvalidation.X <- predict(pre.fit, xvalidation.features)
testing.X <- predict(pre.fit, testing.clean)

if (LOAD_FROM_FILE) {
    load(file='multiple_networks.save')
} else {
    nns <- multiple_train_nn(training.Y, training.X, 5, 39, 2, stepmax=1e+06)
}
```

Let see how the generalization error behaves with the network complexity.



We can see that the best geometry is about 35 units, adding more units to the network perhaps it would reduce the cross validation error even more but it would take a lot of time, as it is being trained with only one core.

```
nn <- nns$`35`
predicted <- predict_nn(nn, testing.X)
```

```
knitr::kable(res, caption = "Test results")
```

Table 1: Test results

test	predicted
1	B
2	C
3	B
4	A
5	B
6	A
7	A
8	A
9	C
10	B
11	B
12	B
13	A

test	predicted
14	C
15	E
16	C
17	A
18	A
19	B
20	B