ML-05-AlGhammari-Schmidt

Bernd Schmidt , Osamah Al-Ghammari

November 03, 2017

```
require(plyr)
require(lattice)
require(corrplot)
require(caret)
require(doMC)
registerDoMC(3)
library(zoo)
```

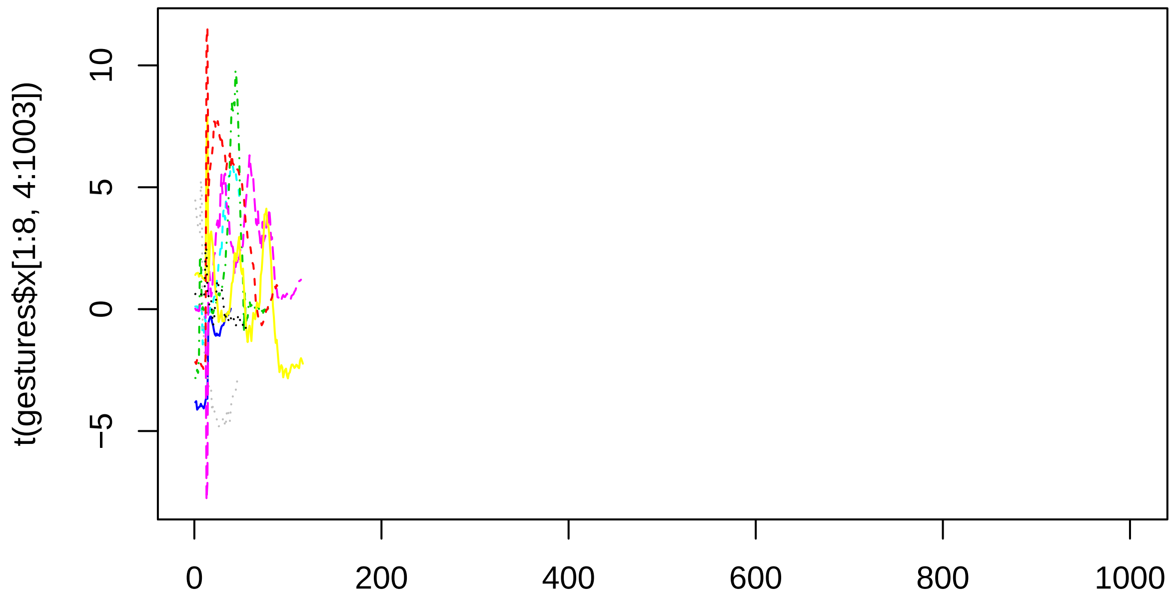
Gesture Recognition

```
filedir <- "../dat"

# get all filenames
filenames <- list.files(filedir, full.names = T, pattern="*.csv")

gestures <- list()

gestures$x <- ldply(filenames[1], read.table, sep=',', fill = T, col.names = c('gesture', 'person', 'sam
gestures$y <- ldply(filenames[2], read.table, sep=',', fill = T, col.names = c('gesture', 'person', 'sam
gestures$z <- ldply(filenames[3], read.table, sep=',', fill = T, col.names = c('gesture', 'person', 'sam
```

Visualization

Currently the samples have different lengths and there are **NA** at the start and/or the end of the sample.

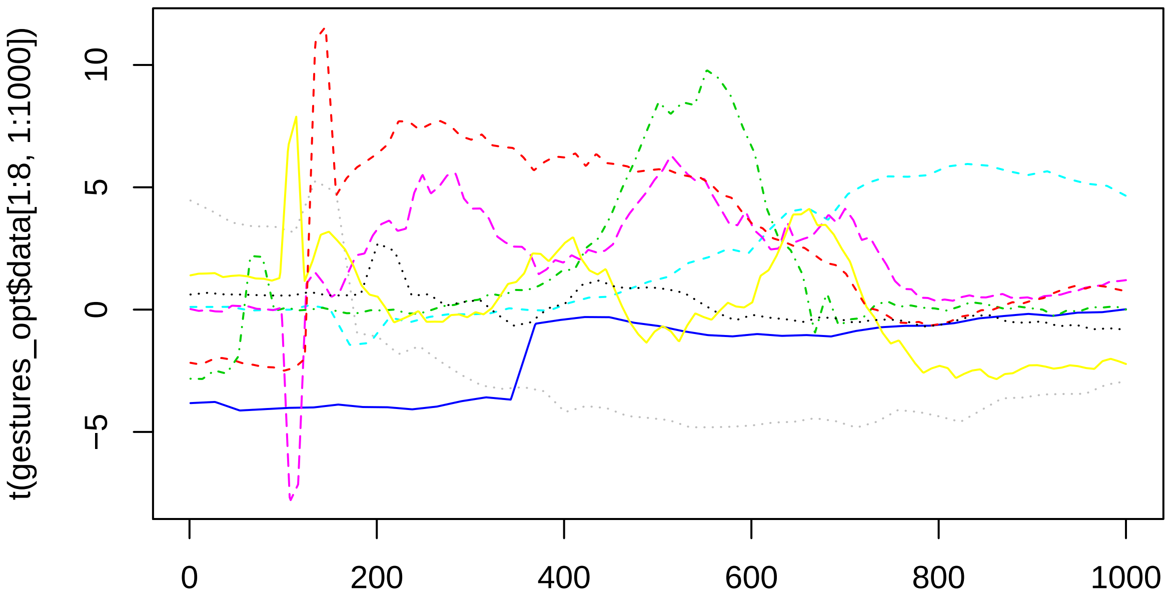
```
matplot(t(gestures$x[1:8,4:1003]), type='l', col=factor(gestures$x[1:8,1]))
```



Optimization

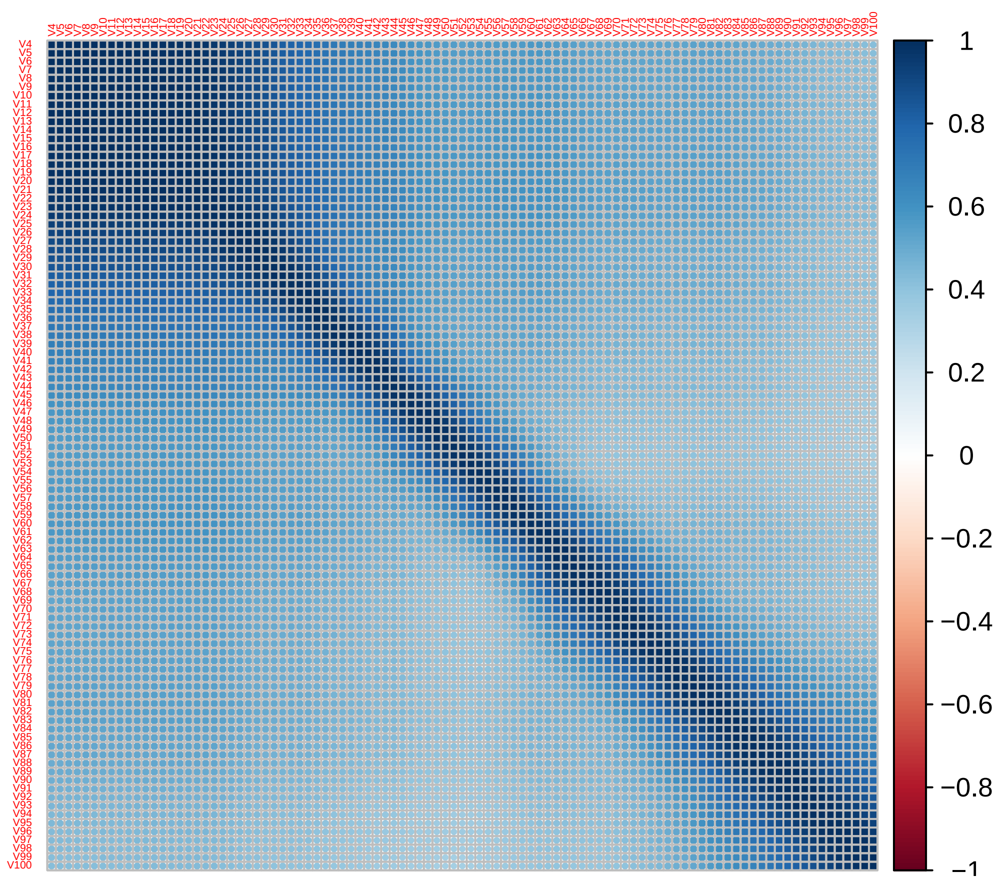
For optimization, all **NA** values are removed and the values are interpolated to 1000 values per sample. After that, a rolling median is applied to the sample to smooth it.

```
stepwidth <- 1/1000
optimize <- function(r) {
  row <- r[!is.na(r)] # remove all NA values
  row_approx <- approx(x = seq(0,1,1/(length(row[4:length(row)])-1)), y = row[4:length(row)], xout = se
  #row_runmed <- as.numeric(runmed(row_approx, k = 11)) # filter
  rollapply(row_approx, 30, median, na.rm=T)
  row_approx[1:1000]
}
gestures_opt <- list()
gestures_opt$x <- as.data.frame(t(apply(gestures$x, 1, optimize)))
gestures_opt$y <- as.data.frame(t(apply(gestures$y, 1, optimize)))
gestures_opt$z <- as.data.frame(t(apply(gestures$z, 1, optimize)))

gestures_opt$gesture <- gestures$x[,1]
gestures_opt$data <- gestures_opt$x
gestures_opt$data[,1001:2000] <- gestures_opt$y
gestures_opt$data[,2001:3000] <- gestures_opt$z
matplot(t(gestures_opt$data[1:8,1:1000]), type='l', col=factor(gestures_opt$gesture[1:8]))
```

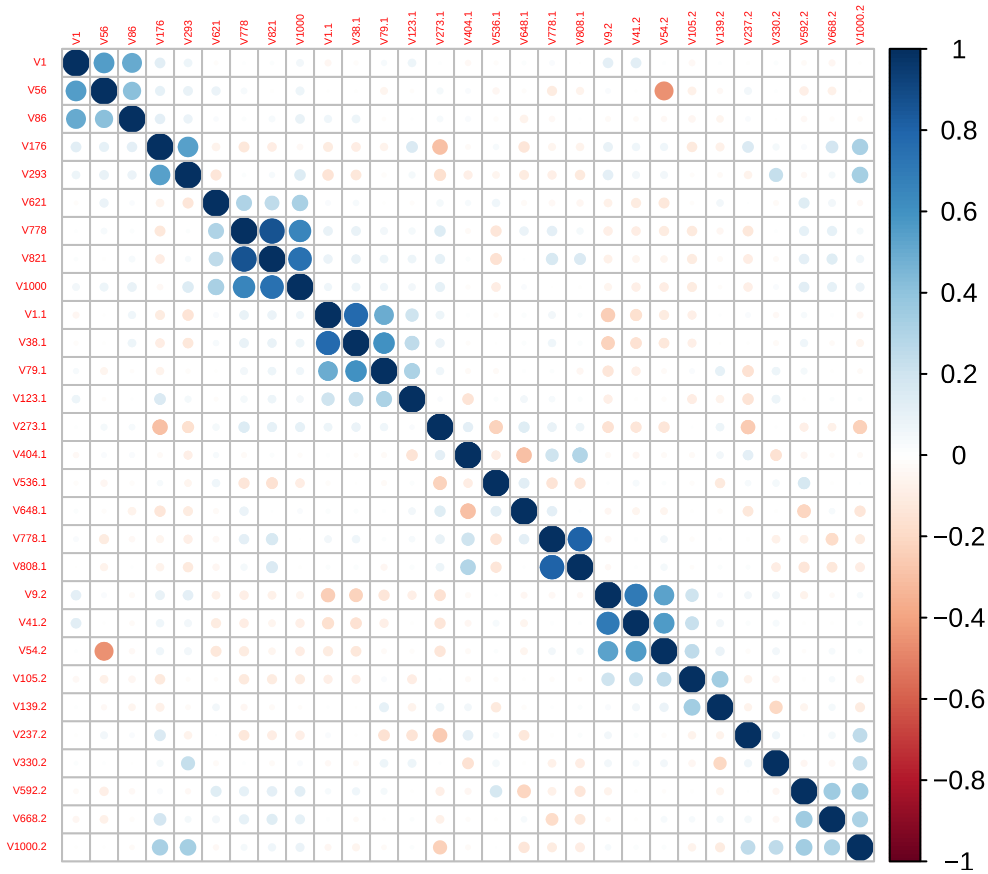


Data Validation and Optimization

```r
# feature correlation as plot
corrplot(cor(gestures_opt$data[,4:100]), tl.cex = 0.3) # addgrid.col = NA
```



```
# remove correlated variable using ?findCorrelation
foundCorIndexes <- findCorrelation(cor(gestures_opt$data))
#foundCorIndexes
corrplot(cor(gestures_opt$data[,-foundCorIndexes]), tl.cex = 0.3)
```




```
# remove the features from the data
gestures_opt$data <- gestures_opt$data[,-foundCorIndexes]
```

Data Partitioning

```
# split into training and test data
set.seed(1704)
indexes_train <- createDataPartition(gestures_opt$gesture, p=0.75, list = F)
indexes_test <- (1:nrow(gestures_opt$data))[-indexes_train]

training <- gestures_opt$data[indexes_train,]
training_gest <- gestures_opt$gesture[indexes_train]
testing <- gestures_opt$data[indexes_test,]
testing_gest <- gestures_opt$gesture[indexes_test]
```

Feature Selection

```r
sbfRes <- sbf(x = training, y = training_gest, sbfControl = sbfControl(functions = rfSBF, method = 'repe
sbfRes
```

```
##
## Selection By Filter
##
## Outer resampling method: Cross-Validated (10 fold, repeated 5 times)
##
## Resampling performance:
##
##  Accuracy  Kappa AccuracySD KappaSD
##    0.9324 0.9228    0.02106 0.02408
##
## Using the training set, 29 variables were selected:
##    V1, V56, V86, V176, V293...
##
## During resampling, the top 5 selected variables (out of a possible 29):
##    V1 (100%), V1.1 (100%), V1000 (100%), V1000.2 (100%), V105.2 (100%)
##
## On average, 29 variables were selected (min = 29, max = 29)
```

```r
sbfRes$optVariables
```

```
##  [1] "V1"      "V56"     "V86"     "V176"    "V293"    "V621"    "V778"
##  [8] "V821"    "V1000"   "V1.1"    "V38.1"   "V79.1"   "V123.1"  "V273.1"
## [15] "V404.1"  "V536.1"  "V648.1"  "V778.1"  "V808.1"  "V9.2"    "V41.2"
## [22] "V54.2"   "V105.2"  "V139.2"  "V237.2"  "V330.2"  "V592.2"  "V668.2"
## [29] "V1000.2"
```

```r
gestures_opt$data <- gestures_opt$data[,sbfRes$optVariables]
```

Model Training

Now we can use this data to train a model for detecting the gestures

```r
models <- list()
```

```r
trControl <- trainControl(
    method = 'repeatedcv', # none, cv, repeatedcv, LOOCV, ...
    number = 10, # nr of CV partitions
    repeats = 20, # nr of partitioning repetitions
    returnData = F,
    # classProbs = T, # enable computation of class probabilities?
    # summaryFunction = twoClassSummary, # use when classifying two classes
    returnResamp = 'final', # return CV partition results for best model
    allowParallel = T
)

#trControl <- trainControl(
#                  method = 'LOOCV',
#                  preProcOptions = list(thresh = 0.9),
#                  returnResamp = 'final',
#                  returnData = F,
#                  savePredictions = T,
```

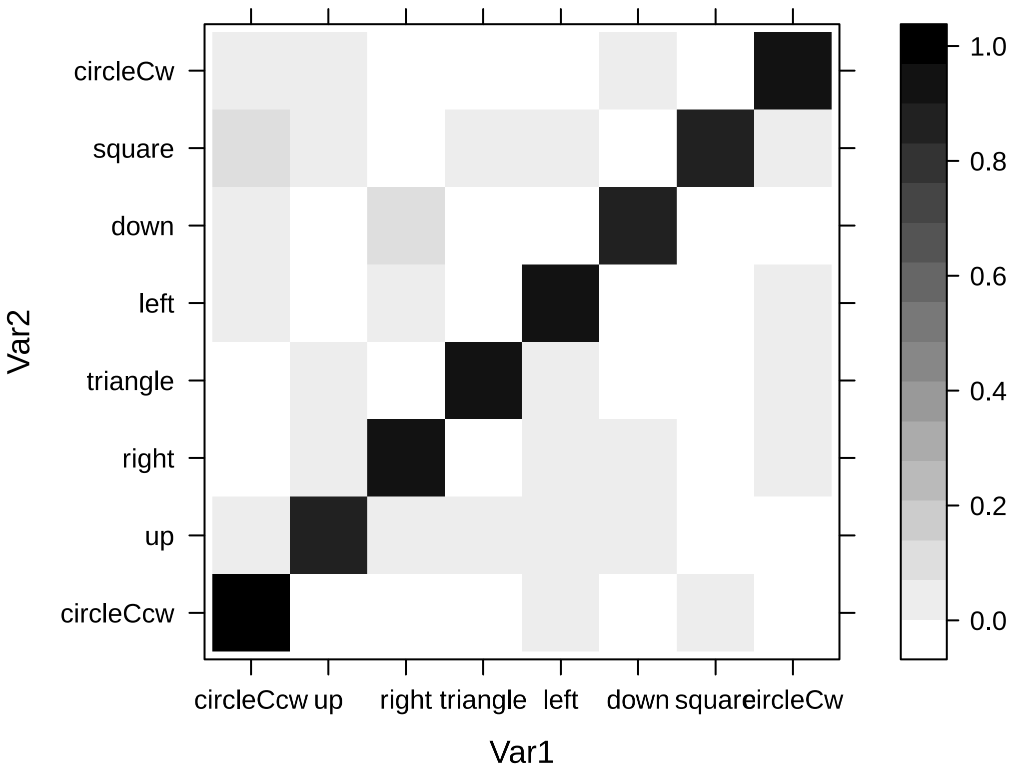
```
#                  allowParallel = T
#                )
```

KNN

```
models$knn <- train(training,
               factor(training_gest),
               method = 'knn',
               preProcess = c('center', 'scale', 'pca'),
               metric = 'Kappa',
               trControl = trControl
               )
models$knn
```

```
## k-Nearest Neighbors
##
## Pre-processing: centered (29), scaled (29), principal component
##  signal extraction (29)
## Resampling: Cross-Validated (10 fold, repeated 20 times)
## Summary of sample sizes: 1461, 1463, 1461, 1461, 1463, 1461, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.9101278  0.8972836
##   7  0.9043109  0.8906346
##   9  0.8960593  0.8812033
##
## Kappa was used to select the optimal model using  the largest value.
## The final value used for the model was k = 5.
```

```
predicted <- predict(models$knn, newdata = testing)

# to ensure, that also when one level is not predicted, the results can be displayed
u = union(predicted, testing_gest)
t = table(factor(predicted, u), factor(testing_gest, u))
conf <- confusionMatrix(t)

levelplot(sweep(conf$table, MARGIN = 2, STATS = colSums(conf$table), FUN = `/`), col.regions = gray(100
```



LDA

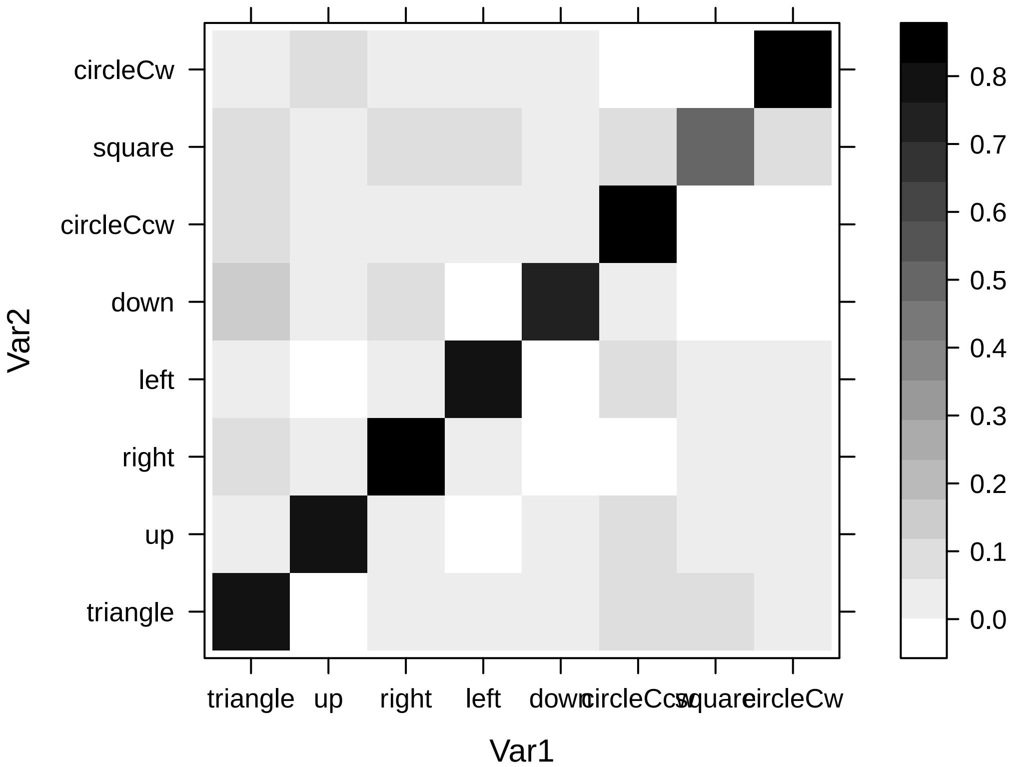
To compare the results, now a *lda* model with the same parameters is trained.

```
models$lda <- train(training,
               factor(training_gest),
               method = 'lda',
               preProcess = c('center', 'scale', 'pca'),
               metric = 'Kappa',
               trControl = trControl
               )
models$lda
```

```
## Linear Discriminant Analysis
##
## Pre-processing: centered (29), scaled (29), principal component
##  signal extraction (29)
## Resampling: Cross-Validated (10 fold, repeated 20 times)
## Summary of sample sizes: 1464, 1459, 1463, 1460, 1463, 1462, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7525714  0.717213
```

```
predicted <- predict(models$lda, newdata = testing)

# to ensure, that also when one level is not predicted, the results can be displayed
u = union(predicted, testing_gest)
t = table(factor(predicted, u), factor(testing_gest, u))
```

```r
conf <- confusionMatrix(t)

levelplot(sweep(conf$table, MARGIN = 2, STATS = colSums(conf$table), FUN = `/`), col.regions = gray(100
```



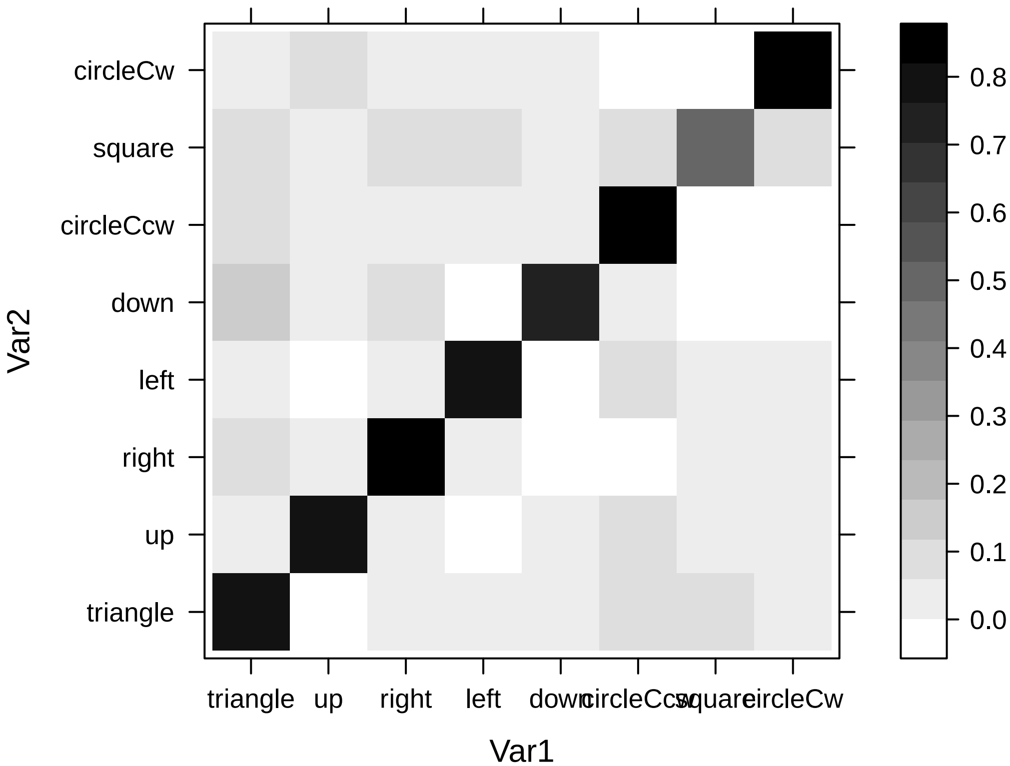
LDA2

```r
models$lda2 <- train(training,
                factor(training_gest),
                method = 'lda2',
                preProcess = c('center', 'scale', 'pca'),
                metric = 'Kappa',
                trControl = trControl
                )
models$lda2
```

```
## Linear Discriminant Analysis
##
## Pre-processing: centered (29), scaled (29), principal component
##  signal extraction (29)
## Resampling: Cross-Validated (10 fold, repeated 20 times)
## Summary of sample sizes: 1463, 1462, 1461, 1461, 1461, 1460, ...
## Resampling results across tuning parameters:
##
##   dimen  Accuracy   Kappa
##   1      0.3178500  0.2204863
##   2      0.5503663  0.4861023
##   3      0.6504609  0.6005053
##   4      0.7193675  0.6792630
```

```
##   5      0.7422275  0.7053901
##   6      0.7439786  0.7073935
##   7      0.7520484  0.7166168
##
## Kappa was used to select the optimal model using  the largest value.
## The final value used for the model was dimen = 7.
```

```
predicted <- predict(models$lda2, newdata = testing)

# to ensure, that also when one level is not predicted, the results can be displayed
u = union(predicted, testing$pers)
t = table(factor(predicted, u), factor(testing_gest, u))
conf <- confusionMatrix(t)

levelplot(sweep(conf$table, MARGIN = 2, STATS = colSums(conf$table), FUN = `/`), col.regions = gray(100
```



Neural Network

```
models$nn <- train(x = training,
           y = training_gest, # only use train data here!
           method = 'nnet',
           metric = 'Kappa',
           tuneGrid = expand.grid(size=9:17, decay=3**(-1:3)), # nnet parameters: size = #neurons in hi
           trControl = trControl
)
```

```
## # weights:  654
## initial  value 3990.142022
## iter  10 value 1505.940112
```

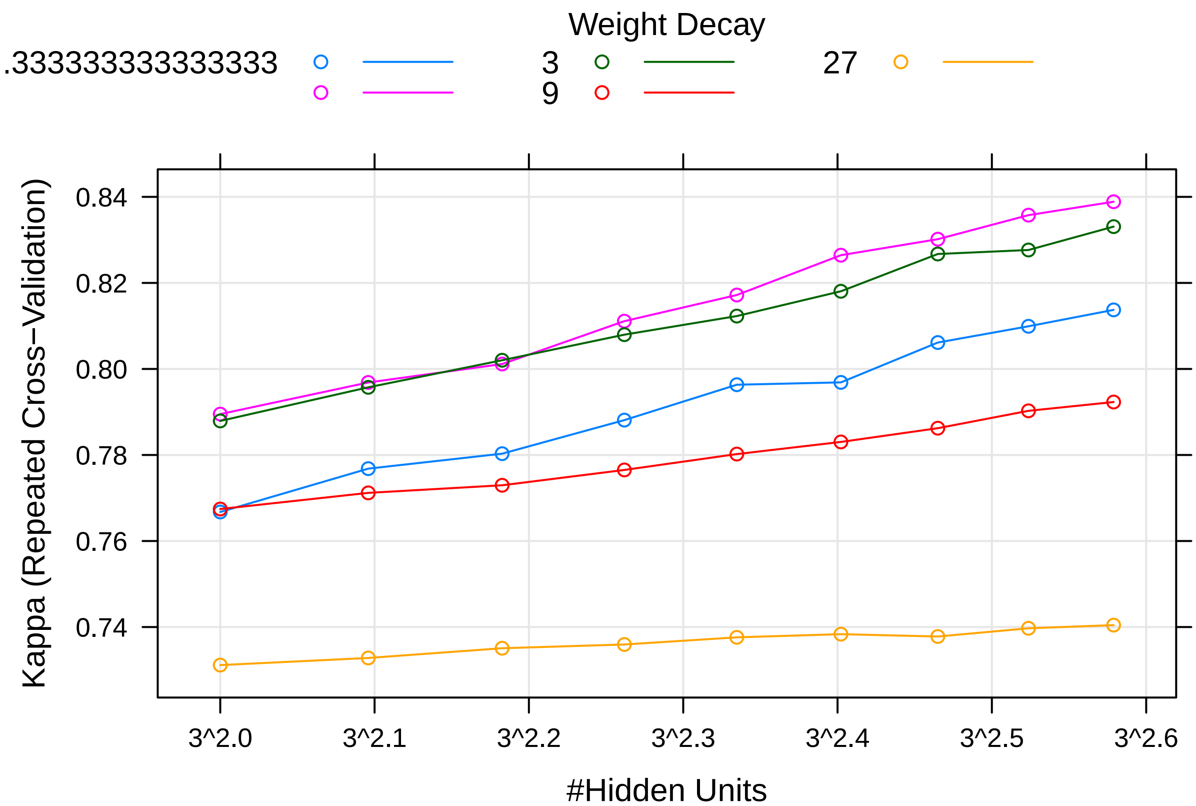
```
## iter  20 value 1196.080184
## iter  30 value 1042.133598
## iter  40 value 932.459517
## iter  50 value 864.476085
## iter  60 value 812.417176
## iter  70 value 778.681123
## iter  80 value 761.173288
## iter  90 value 751.637898
## iter 100 value 744.686764
## final  value 744.686764
## stopped after 100 iterations
```

```r
print(models$nn)
```

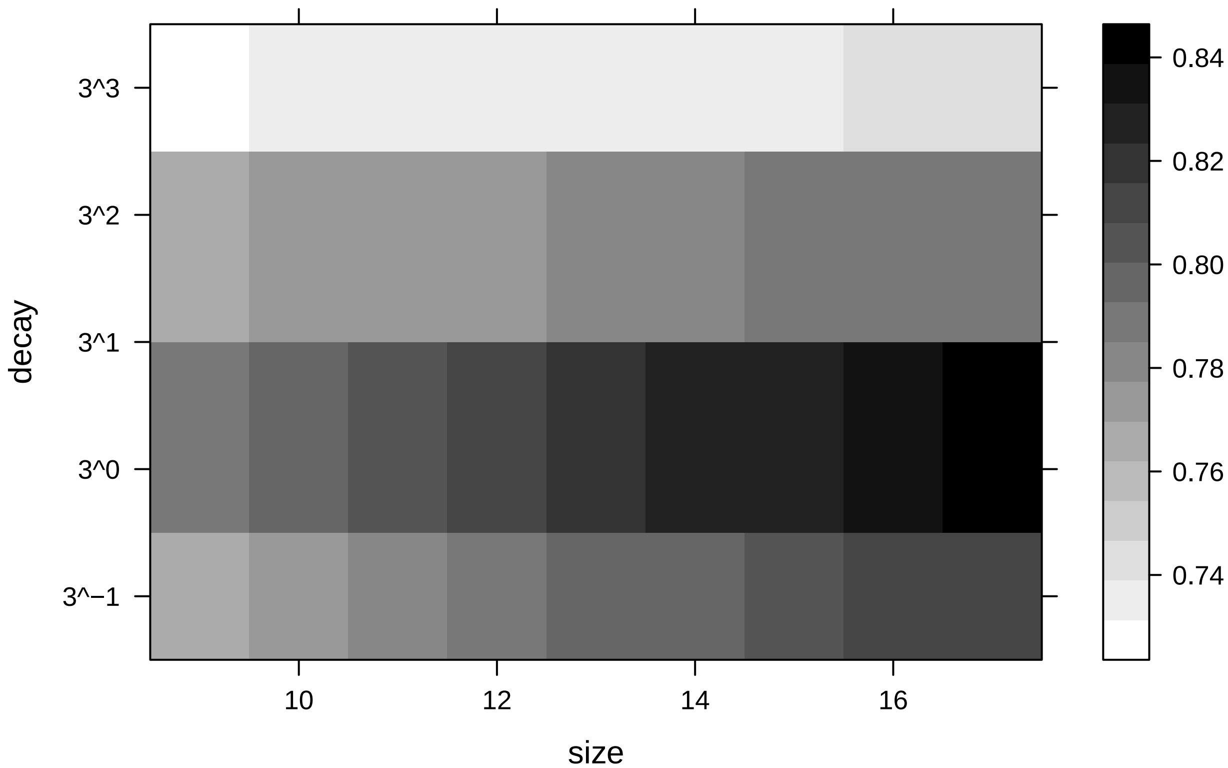
```
## Neural Network
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 20 times)
## Summary of sample sizes: 1459, 1464, 1462, 1459, 1462, 1463, ...
## Resampling results across tuning parameters:
##
##   size  decay       Accuracy   Kappa
##    9     0.3333333  0.7959168  0.7667516
##    9     1.0000000  0.8158232  0.7895013
##    9     3.0000000  0.8144319  0.7879141
##    9     9.0000000  0.7965145  0.7674349
##    9    27.0000000  0.7647713  0.7311543
##   10     0.3333333  0.8047262  0.7768216
##   10     1.0000000  0.8222521  0.7968504
##   10     3.0000000  0.8212712  0.7957319
##   10     9.0000000  0.7998023  0.7711912
##   10    27.0000000  0.7662236  0.7328131
##   11     0.3333333  0.8077746  0.7803035
##   11     1.0000000  0.8260074  0.8011440
##   11     3.0000000  0.8267741  0.8020183
##   11     9.0000000  0.8013464  0.7729551
##   11    27.0000000  0.7682029  0.7350765
##   12     0.3333333  0.8146005  0.7881046
##   12     1.0000000  0.8347241  0.8111049
##   12     3.0000000  0.8319824  0.8079698
##   12     9.0000000  0.8044623  0.7765176
##   12    27.0000000  0.7689612  0.7359435
##   13     0.3333333  0.8217962  0.7963297
##   13     1.0000000  0.8400491  0.8171886
##   13     3.0000000  0.8357666  0.8122946
##   13     9.0000000  0.8076897  0.7802071
##   13    27.0000000  0.7704153  0.7376064
##   14     0.3333333  0.8222788  0.7968804
##   14     1.0000000  0.8481415  0.8264401
##   14     3.0000000  0.8408076  0.8180578
##   14     9.0000000  0.8101637  0.7830326
##   14    27.0000000  0.7710600  0.7383429
##   15     0.3333333  0.8303809  0.8061385
##   15     1.0000000  0.8514034  0.8301669
##   15     3.0000000  0.8483891  0.8267225
```

```
##    15      9.0000000  0.8129588  0.7862276
##    15     27.0000000  0.7705861  0.7377998
##    16      0.3333333  0.8336836  0.8099143
##    16      1.0000000  0.8562921  0.8357546
##    16      3.0000000  0.8492037  0.8276525
##    16      9.0000000  0.8164917  0.7902656
##    16     27.0000000  0.7722553  0.7397080
##    17      0.3333333  0.8370254  0.8137345
##    17      1.0000000  0.8590212  0.8388752
##    17      3.0000000  0.8539551  0.8330853
##    17      9.0000000  0.8182819  0.7923131
##    17     27.0000000  0.7729018  0.7404476
##
## Kappa was used to select the optimal model using  the largest value.
## The final values used for the model were size = 17 and decay = 1.
```

```
plot(models$nn, scales = list(x = list(log = 3)))
```




```
levelplot(x = Kappa ~ size * decay, data = models$nn$results[models$nn$results$decay!=3 & models$nn$resu
```




```r
# nnet plots: https://beckmw.wordpress.com/2013/11/14/visualizing-neural-networks-in-r-update/
library(devtools)
source_url('https://gist.githubusercontent.com/fawda123/7471137/raw/466c1474d0a505ff044412703516c34f1a4
```

```
## SHA-1 hash of file is 74c80bd5ddbc17ab3ae5ece9c0ed9beb612e87ef
```

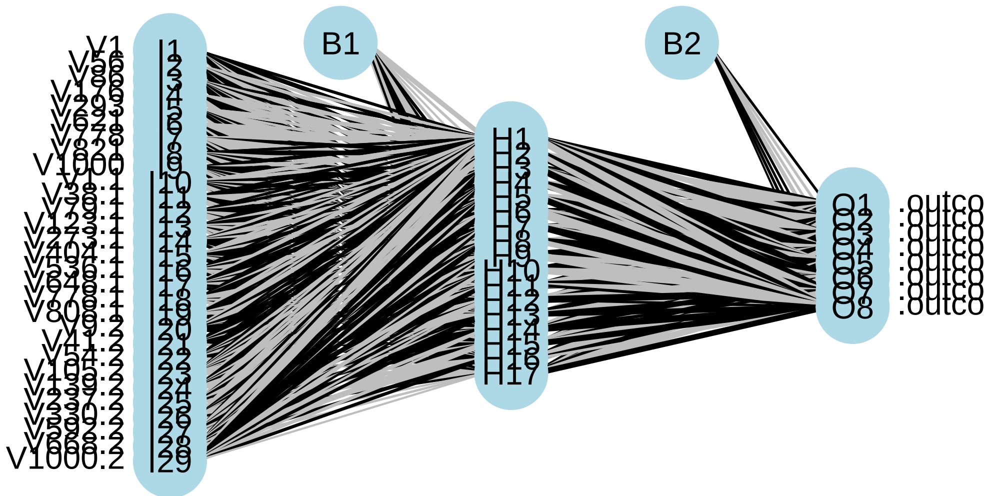
```r
plot.nnet(models$nn$finalModel)
```

```
## Loading required package: scales
```

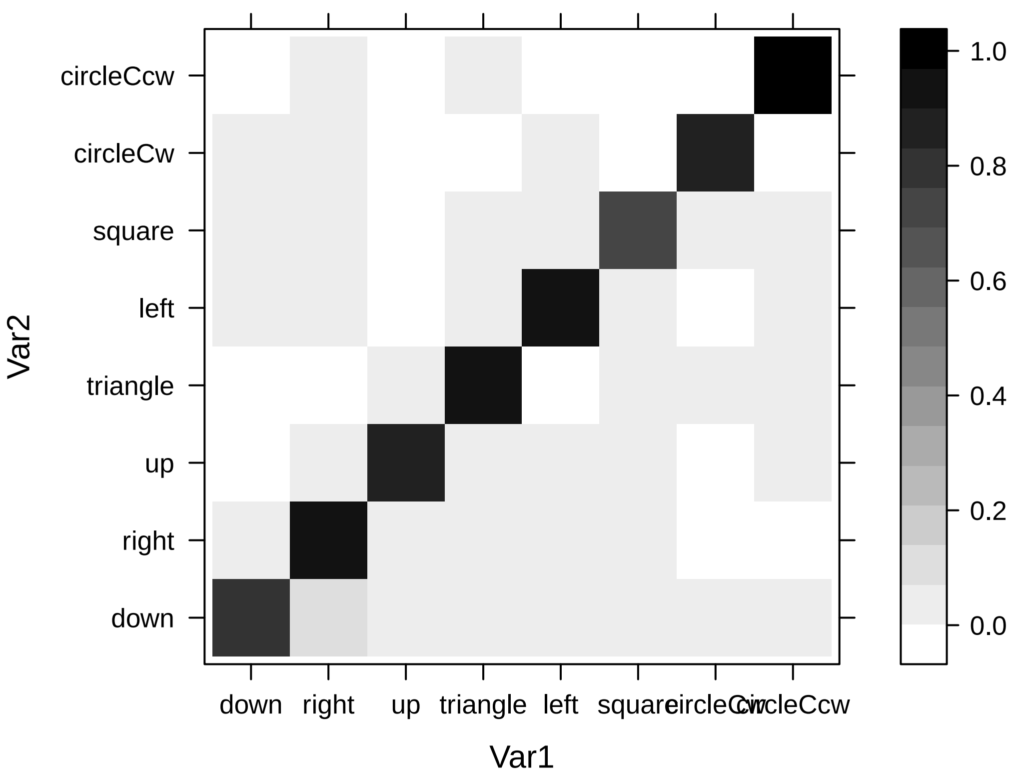
```
## Loading required package: reshape
```

```
##
## Attaching package: 'reshape'
```

```
## The following objects are masked from 'package:plyr':
##
##     rename, round_any
```




```r
predicted <- predict(models$nn, newdata = testing)

# to ensure, that also when one level is not predicted, the results can be displayed
u = union(predicted, testing$pers)
t = table(factor(predicted, u), factor(testing_gest, u))
conf <- confusionMatrix(t)

levelplot(sweep(conf$table, MARGIN = 2, STATS = colSums(conf$table), FUN = `/`), col.regions = gray(100
```



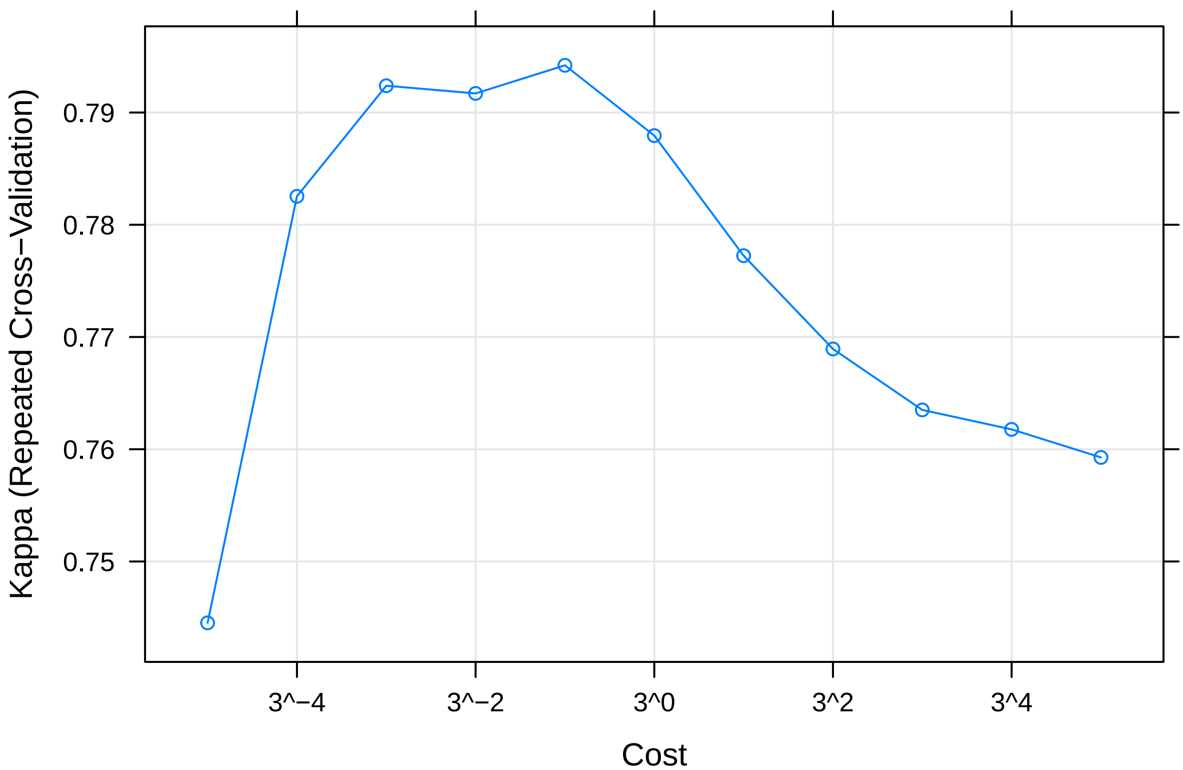
SVM

```
train_model <- function(method, tuneGrid=NULL) {
  train(x = training, # in real life apps only use train data here!
        y = training_gest, # in real life apps only use train data here!
        method = method,
        metric = 'Kappa',
        tuneGrid = tuneGrid,
        trControl = trControl
  )
}
models$svmLinear <- train_model('svmLinear', tuneGrid = expand.grid(C=3**(-5:5)))
```

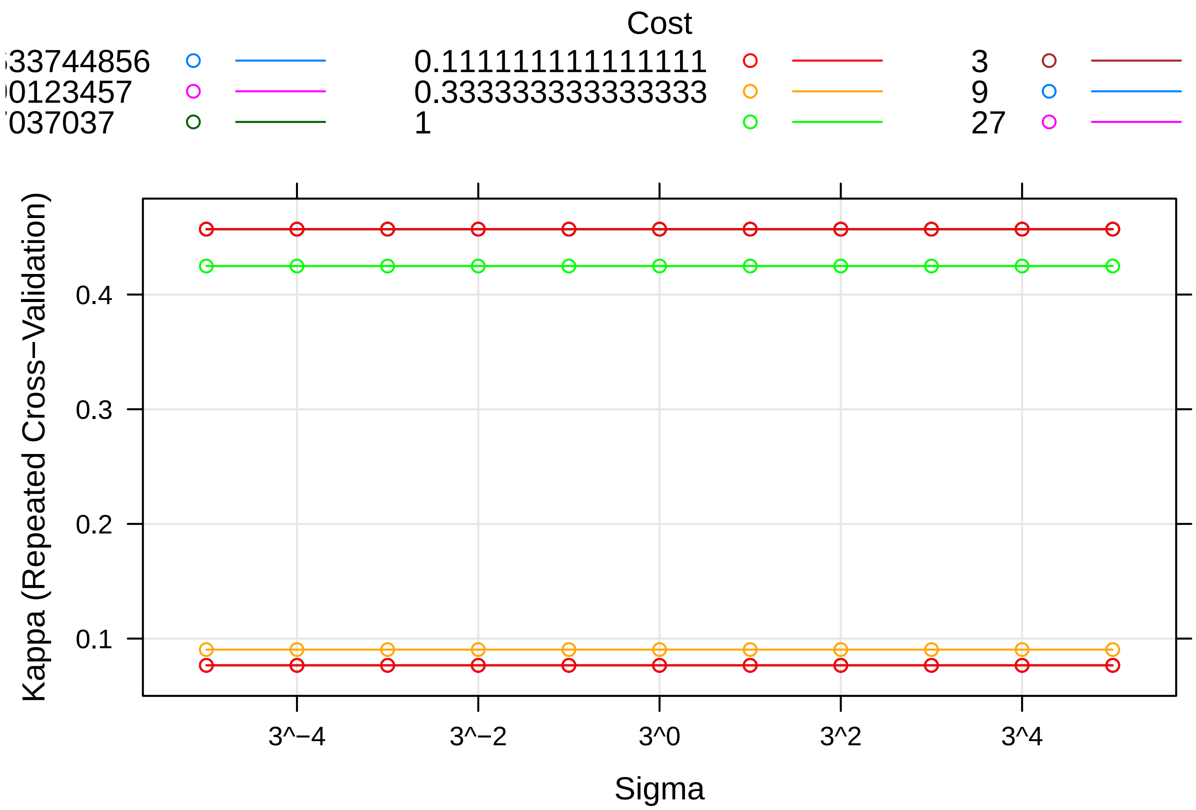
```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:scales':
##
##     alpha

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
models$svmRadial <- train_model('svmRadial', tuneGrid = expand.grid(C=3**(-5:5), sigma=3**(-5:5)))
```

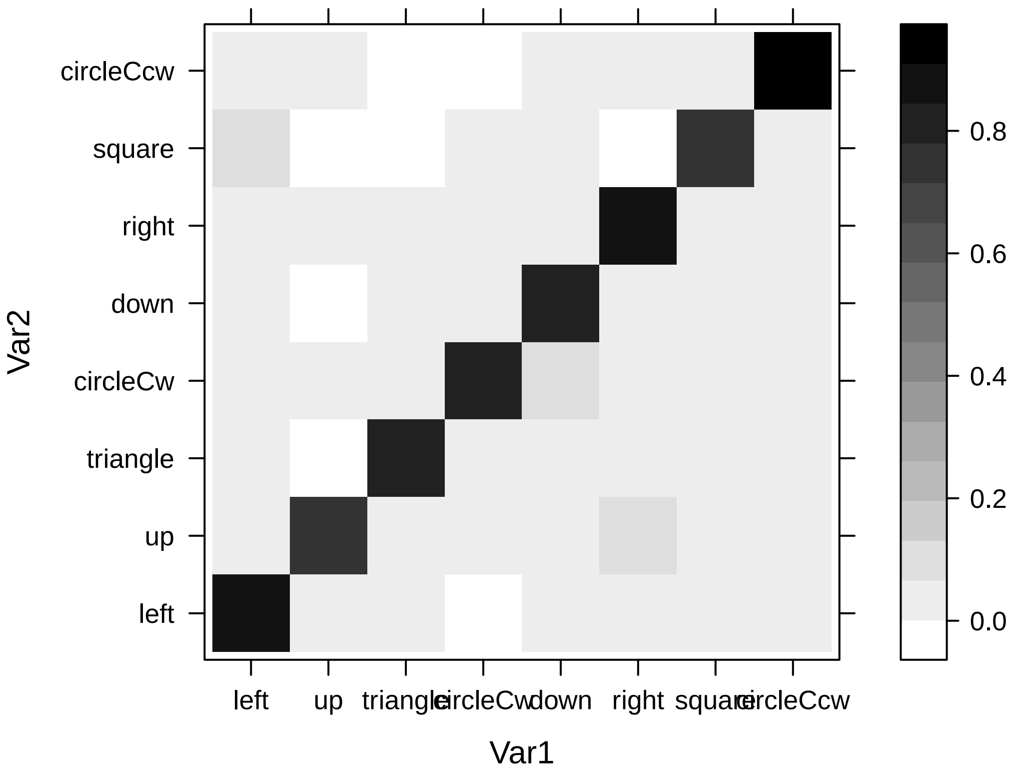
```
print(plot(models$svmLinear, scales=list(x=list(log=3))))
```



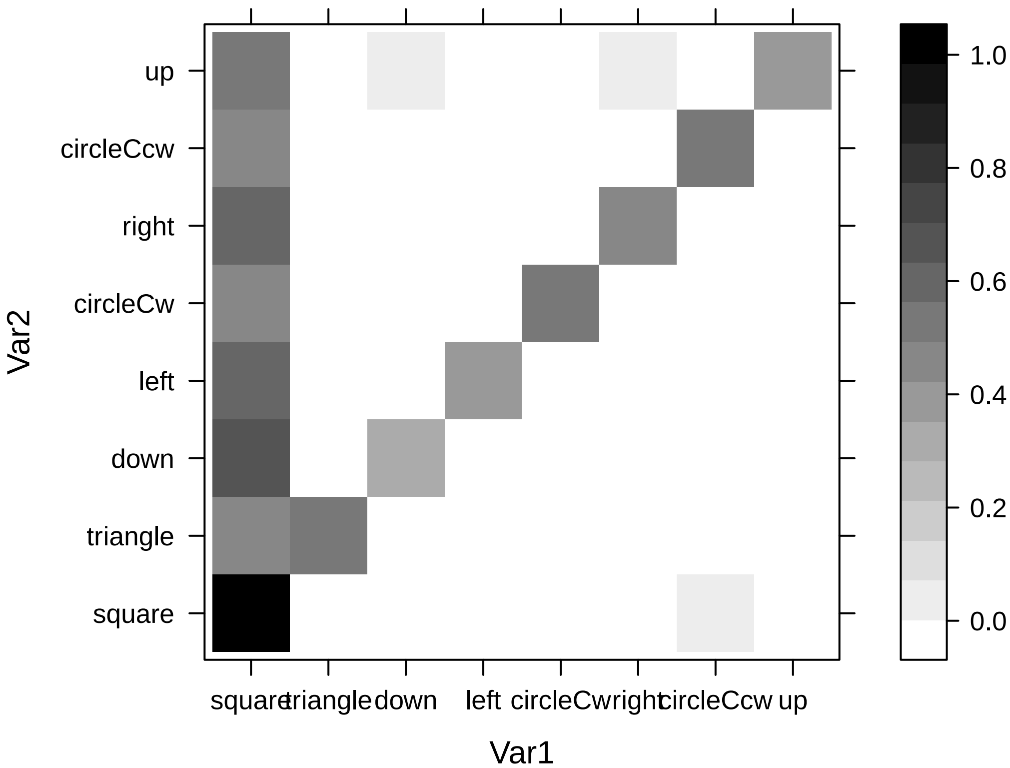

```
print(plot(models$svmRadial, scales=list(x=list(log=3))))
```




```r
predicted <- predict(models$svmLinear, newdata = testing)

# to ensure, that also when one level is not predicted, the results can be displayed
u = union(predicted, testing$pers)
t = table(factor(predicted, u), factor(testing_gest, u))
conf <- confusionMatrix(t)

levelplot(sweep(conf$table, MARGIN = 2, STATS = colSums(conf$table), FUN = `/`), col.regions = gray(100
```




```
predicted <- predict(models$svmRadial, newdata = testing)

# to ensure, that also when one level is not predicted, the results can be displayed
u = union(predicted, testing$pers)
t = table(factor(predicted, u), factor(testing_gest, u))
conf <- confusionMatrix(t)

levelplot(sweep(conf$table, MARGIN = 2, STATS = colSums(conf$table), FUN = `/`), col.regions = gray(100
```



Result Comparison

```
# save models to file to ensure that the results were not lost
saveRDS(object = models, file = "gesture_models.RDS")
results <- resamples(models)
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: knn, lda, lda2, nn, svmLinear, svmRadial
## Number of resamples: 200
##
## Accuracy
##                Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## knn       0.8447205 0.8961825 0.9130435 0.9101278 0.9259259 0.9562500    0
## lda       0.6625767 0.7325019 0.7530864 0.7525714 0.7730061 0.8271605    0
## lda2      0.6687117 0.7271316 0.7500000 0.7520484 0.7743902 0.8414634    0
## nn        0.7852761 0.8447205 0.8588957 0.8590212 0.8765432 0.9254658    0
## svmLinear 0.7267081 0.8024691 0.8220859 0.8199357 0.8395062 0.8834356    0
## svmRadial 0.4320988 0.4992424 0.5279503 0.5249349 0.5533158 0.6073620    0
##
## Kappa
##                Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## knn       0.8225622 0.8813402 0.9006085 0.8972836 0.9153439 0.9500000    0
## lda       0.6144749 0.6941882 0.7177637 0.7172130 0.7404989 0.8024304    0
## lda2      0.6212565 0.6881334 0.7142857 0.7166168 0.7420945 0.8187999    0
```

```
## nn        0.7545603 0.8225230 0.8387319 0.8388752 0.8588804 0.9148186    0
## svmLinear 0.6876543 0.7742504 0.7966401 0.7942047 0.8165625 0.8667728    0
## svmRadial 0.3517181 0.4266671 0.4597415 0.4570522 0.4894299 0.5516782    0
```

```
bwplot(results)
```