Support Vector Machines

```
library(mlbench)
library(caret)
library(e1071) # has function called tune.svm and svm function. `train` function in caret is more flexi
```

We use the Pima Indians Diabetes Database for illustration. The data contain 768 observations and 9 variables. The outcome is a binary variable `diabetes`.

```
data(PimaIndiansDiabetes)
dat <- PimaIndiansDiabetes
dat$diabetes <- factor(dat$diabetes, c("pos", "neg"))

set.seed(1)
rowTrain <- createDataPartition(y = dat$diabetes,
                                p = 0.75,
                                list = FALSE)
```

Using e1071

Linear boundary

Most real data sets will not be fully separable by a linear boundary. Support vector classifiers with a tuning parameter `cost`, which quantifies the penalty associated with having an observation on the wrong side of the classification boundary, can be used to build a linear boundary.

```
set.seed(1)
linear.tune <- tune.svm(diabetes~.,
                        data = dat[rowTrain,],
                        kernel = "linear",
                        cost = exp(seq(-5,1,len = 20))) # cost is the tuning parameter. Has to be non-n

# by defaut it does 10 fold cross-validation

summary(linear.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##        cost
##  0.06145355
##
## - best performance: 0.2274955
##
## - Detailed performance results:
##           cost     error dispersion
## 1  0.006737947 0.2431942 0.06750622
## 2  0.009240026 0.2431337 0.06033096
## 3  0.012671232 0.2414398 0.06607744
## 4  0.017376587 0.2379310 0.06531182
```

```
## 5  0.023829235 0.2327284 0.06062636
## 6  0.032678019 0.2292498 0.05881727
## 7  0.044812724 0.2292498 0.05881727
## 8  0.061453549 0.2274955 0.05911725
## 9  0.084273804 0.2309740 0.05890342
## 10 0.115568167 0.2326981 0.05664725
## 11 0.158483425 0.2344525 0.05366248
## 12 0.217334902 0.2344525 0.05178306
## 13 0.298040375 0.2344525 0.05178306
## 14 0.408715141 0.2344525 0.05178306
## 15 0.560488044 0.2362069 0.05441479
## 16 0.768620527 0.2344525 0.05243932
## 17 1.054041243 0.2361766 0.05481172
## 18 1.445450522 0.2361766 0.05481172
## 19 1.982206318 0.2361766 0.05481172
## 20 2.718281828 0.2361766 0.05481172
```

```
best.linear <- linear.tune$best.model
summary(best.linear)
```

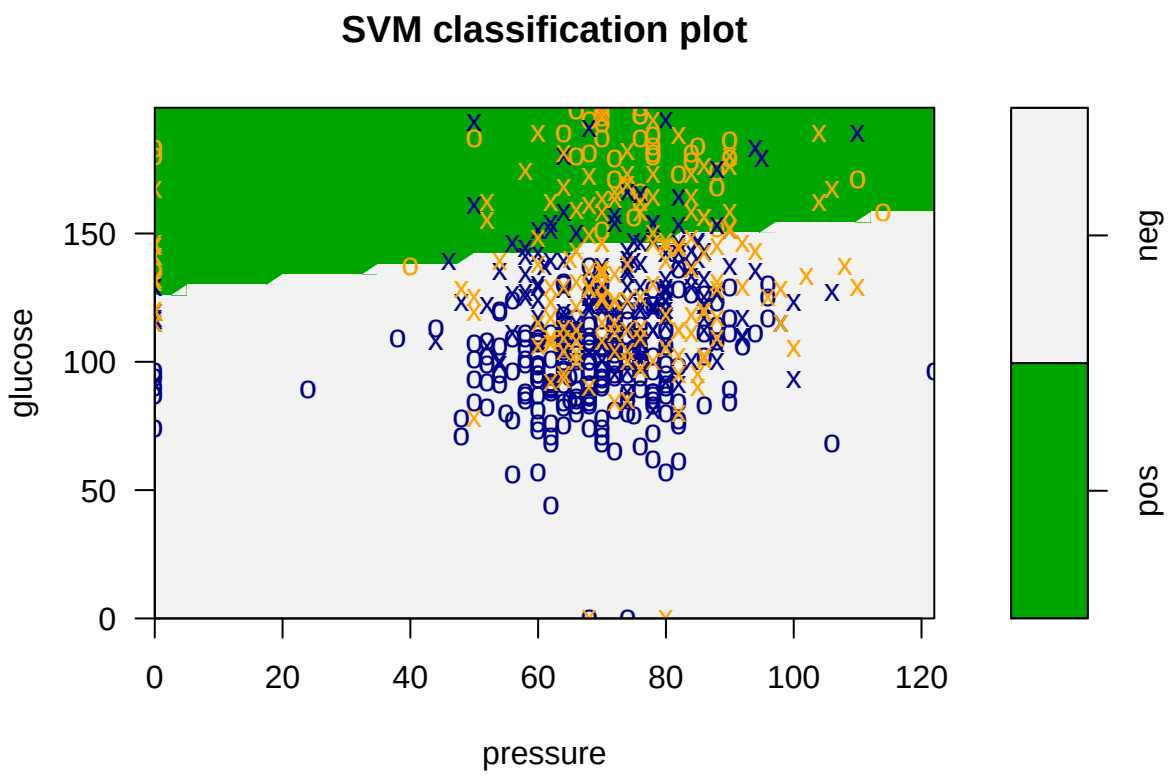
```
##
## Call:
## best.svm(x = diabetes ~ ., data = dat[rowTrain, ], cost = exp(seq(-5,
##     1, len = 20)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.06145355
##       gamma:  0.125
##
## Number of Support Vectors:  323
##
##  ( 162 161 )
##
##
## Number of Classes:  2
##
## Levels:
##  pos neg
```

```
# C classification meaning we are using the tuning with cost. There's anohter called Nu classification.
# they are mostly equivalent.

pred.linear <- predict(best.linear, newdata = dat[-rowTrain,])

confusionMatrix(data = pred.linear,
                reference = dat$diabetes[-rowTrain])
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction pos neg
##        pos  38  16
##        neg  29 109
```

```
##
##                Accuracy : 0.7656
##                  95% CI : (0.6992, 0.8236)
##     No Information Rate : 0.651
##     P-Value [Acc > NIR] : 0.0004037
##
##                   Kappa : 0.4599
##  Mcnemar's Test P-Value : 0.0736383
##
##             Sensitivity : 0.5672
##             Specificity : 0.8720
##          Pos Pred Value : 0.7037
##          Neg Pred Value : 0.7899
##              Prevalence : 0.3490
##          Detection Rate : 0.1979
##    Detection Prevalence : 0.2812
##       Balanced Accuracy : 0.7196
##
##        'Positive' Class : pos
##
```

```r
plot(best.linear, dat[rowTrain,], glucose ~ pressure, # the decision boundary. which two variables we a
     # so in this case we plot glucose and pressure. We need to fix the other six predictors at a const
     # That's what we pass in the slice argument.
     slice = list(pregnant = 5, triceps = 20,
                  insulin = 20, mass = 25,
                  pedigree = 1, age = 50),
                  symbolPalette = c("orange","darkblue"),
                  color.palette = terrain.colors)
```



```
# In the plot, the x = are the support vectors. o = not support vectors. The background shows the decis
```

Radial kernel

In real life the decision boundary might be linear so we can make a non linear boundary as well. Support vector machines can construct classification boundaries that are nonlinear in shape. We use the radial kernel. Here we have 2 tuning parameters.

```
set.seed(1)
radial.tune <- tune.svm(diabetes~.,
                        data = dat[rowTrain,],
                        kernel = "radial",
                        cost = exp(seq(-4,5,len = 10)),
                        gamma = exp(seq(-8,-3,len = 5))) # controns the bandwith of the kernel function

summary(radial.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##       gamma     cost
##  0.00117088 54.59815
##
## - best performance: 0.2292196
##
## - Detailed performance results:
##           gamma         cost     error dispersion
## 1  0.0003354626   0.01831564 0.3490926 0.09596130
## 2  0.0011708796   0.01831564 0.3490926 0.09596130
## 3  0.0040867714   0.01831564 0.3490926 0.09596130
## 4  0.0142642339   0.01831564 0.3490926 0.09596130
## 5  0.0497870684   0.01831564 0.3490926 0.09596130
## 6  0.0003354626   0.04978707 0.3490926 0.09596130
## 7  0.0011708796   0.04978707 0.3490926 0.09596130
## 8  0.0040867714   0.04978707 0.3490926 0.09596130
## 9  0.0142642339   0.04978707 0.3490926 0.09596130
## 10 0.0497870684   0.04978707 0.3456443 0.09977319
## 11 0.0003354626   0.13533528 0.3490926 0.09596130
## 12 0.0011708796   0.13533528 0.3490926 0.09596130
## 13 0.0040867714   0.13533528 0.3490926 0.09596130
## 14 0.0142642339   0.13533528 0.3369328 0.09932768
## 15 0.0497870684   0.13533528 0.2432244 0.07356880
## 16 0.0003354626   0.36787944 0.3490926 0.09596130
## 17 0.0011708796   0.36787944 0.3490926 0.09596130
## 18 0.0040867714   0.36787944 0.3421960 0.09568191
## 19 0.0142642339   0.36787944 0.2414398 0.06948842
## 20 0.0497870684   0.36787944 0.2379008 0.06734931
## 21 0.0003354626   1.00000000 0.3490926 0.09596130
## 22 0.0011708796   1.00000000 0.3422263 0.10659381
## 23 0.0040867714   1.00000000 0.2362069 0.06115266
```

```
## 24 0.0142642339   1.00000000 0.2361766 0.07570632
## 25 0.0497870684   1.00000000 0.2343920 0.07042438
## 26 0.0003354626   2.71828183 0.3491228 0.10014031
## 27 0.0011708796   2.71828183 0.2449183 0.06619431
## 28 0.0040867714   2.71828183 0.2344525 0.06892282
## 29 0.0142642339   2.71828183 0.2343920 0.06763823
## 30 0.0497870684   2.71828183 0.2447973 0.06282148
## 31 0.0003354626   7.38905610 0.2604658 0.06082395
## 32 0.0011708796   7.38905610 0.2379613 0.06278182
## 33 0.0040867714   7.38905610 0.2292498 0.05939587
## 34 0.0142642339   7.38905610 0.2326376 0.05850290
## 35 0.0497870684   7.38905610 0.2484271 0.05791795
## 36 0.0003354626  20.08553692 0.2414398 0.06607744
## 37 0.0011708796  20.08553692 0.2344525 0.05904137
## 38 0.0040867714  20.08553692 0.2326981 0.06111501
## 39 0.0142642339  20.08553692 0.2430732 0.05888892
## 40 0.0497870684  20.08553692 0.2570780 0.05797206
## 41 0.0003354626  54.59815003 0.2292498 0.05881727
## 42 0.0011708796  54.59815003 0.2292196 0.05453336
## 43 0.0040867714  54.59815003 0.2326679 0.06036313
## 44 0.0142642339  54.59815003 0.2518149 0.06618911
## 45 0.0497870684  54.59815003 0.2778887 0.06399190
## 46 0.0003354626 148.41315910 0.2292498 0.05881727
## 47 0.0011708796 148.41315910 0.2378705 0.05922699
## 48 0.0040867714 148.41315910 0.2326376 0.06338075
## 49 0.0142642339 148.41315910 0.2415306 0.07419060
## 50 0.0497870684 148.41315910 0.2934362 0.06062154
```

```
best.radial <- radial.tune$best.model
summary(best.radial)
```

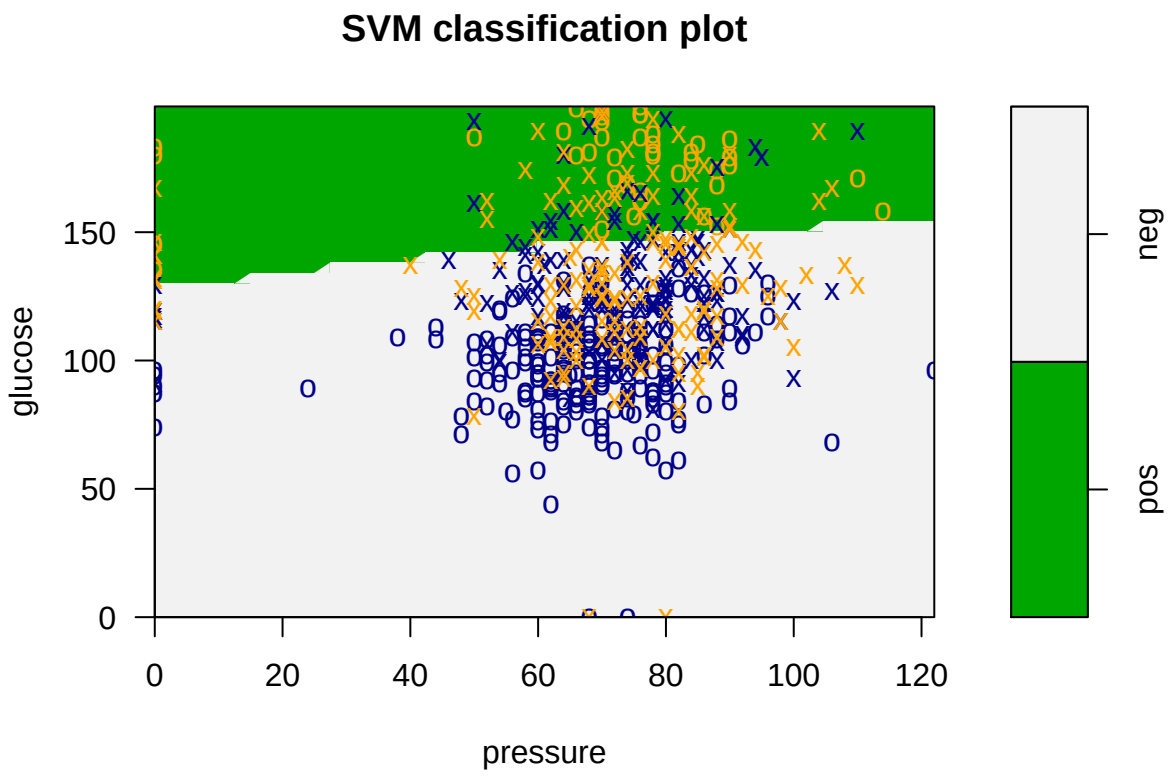
```
##
## Call:
## best.svm(x = diabetes ~ ., data = dat[rowTrain, ], gamma = exp(seq(-8,
##     -3, len = 5)), cost = exp(seq(-4, 5, len = 10)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  54.59815
##       gamma:  0.00117088
##
## Number of Support Vectors:  318
##
##  ( 160 158 )
##
##
## Number of Classes:  2
##
## Levels:
##  pos neg
```

```
pred.radial <- predict(best.radial, newdata = dat[-rowTrain,])
```

```r
confusionMatrix(data = pred.radial,
                reference = dat$diabetes[-rowTrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction pos neg
##        pos  38  15
##        neg  29 110
##
##                Accuracy : 0.7708
##                  95% CI : (0.7048, 0.8283)
##     No Information Rate : 0.651
##     P-Value [Acc > NIR] : 0.0002216
##
##                   Kappa : 0.4699
##  Mcnemar's Test P-Value : 0.0500164
##
##             Sensitivity : 0.5672
##             Specificity : 0.8800
##          Pos Pred Value : 0.7170
##          Neg Pred Value : 0.7914
##              Prevalence : 0.3490
##          Detection Rate : 0.1979
##    Detection Prevalence : 0.2760
##       Balanced Accuracy : 0.7236
##
##        'Positive' Class : pos
##
```

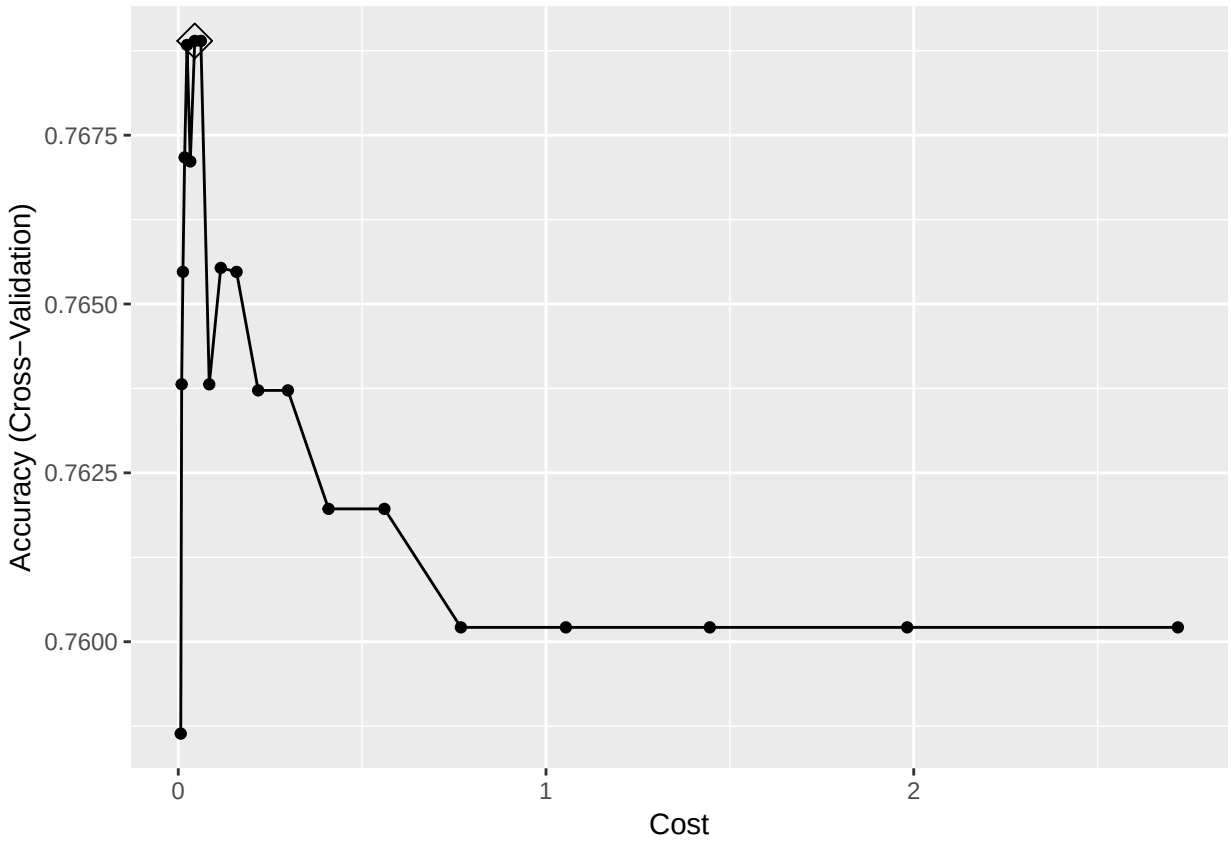
```r
plot(best.radial, dat[rowTrain,], glucose~pressure,
     slice = list(pregnant = 5, triceps = 20,
                  insulin = 20, mass = 25,
                  pedigree = 1, age = 40),
     symbolPalette = c("orange","darkblue"),
     color.palette = terrain.colors)
```



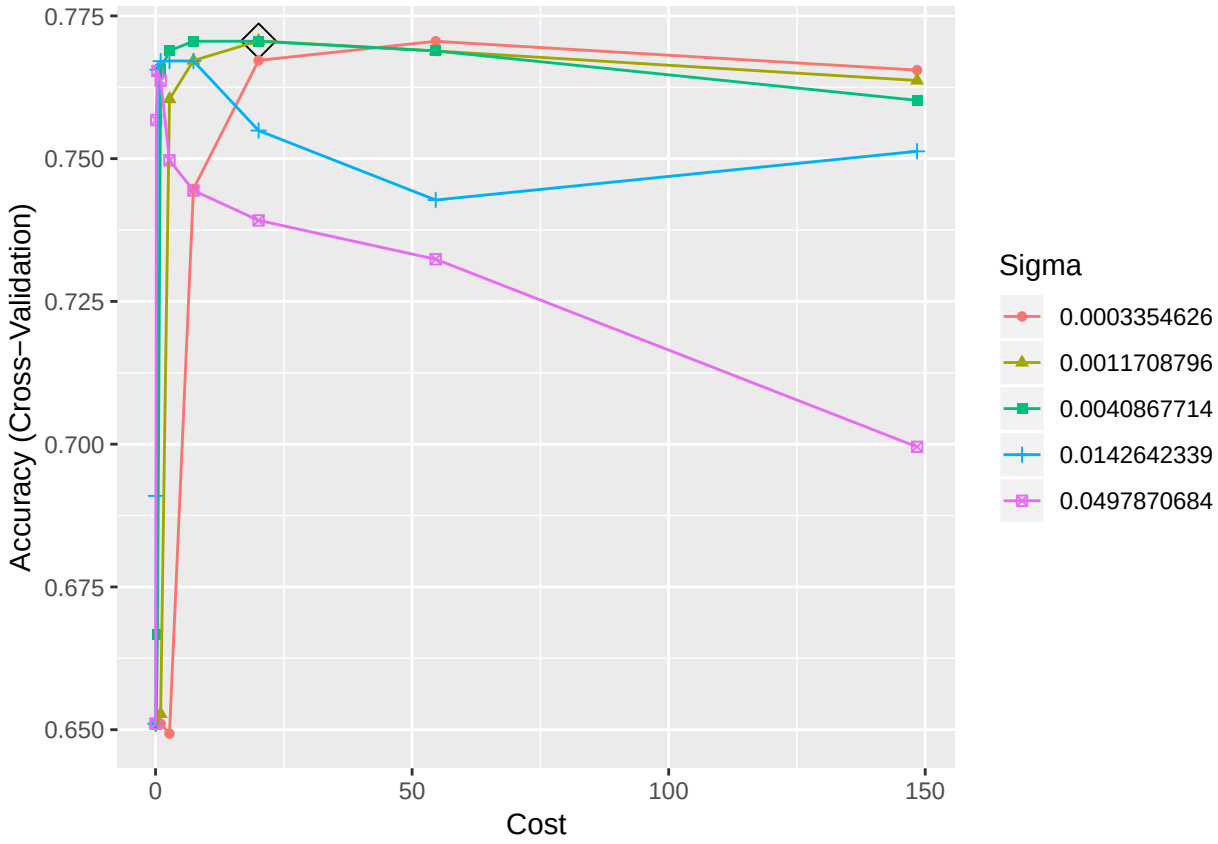
Using `caret`

Caret is recommended. It is more flexible

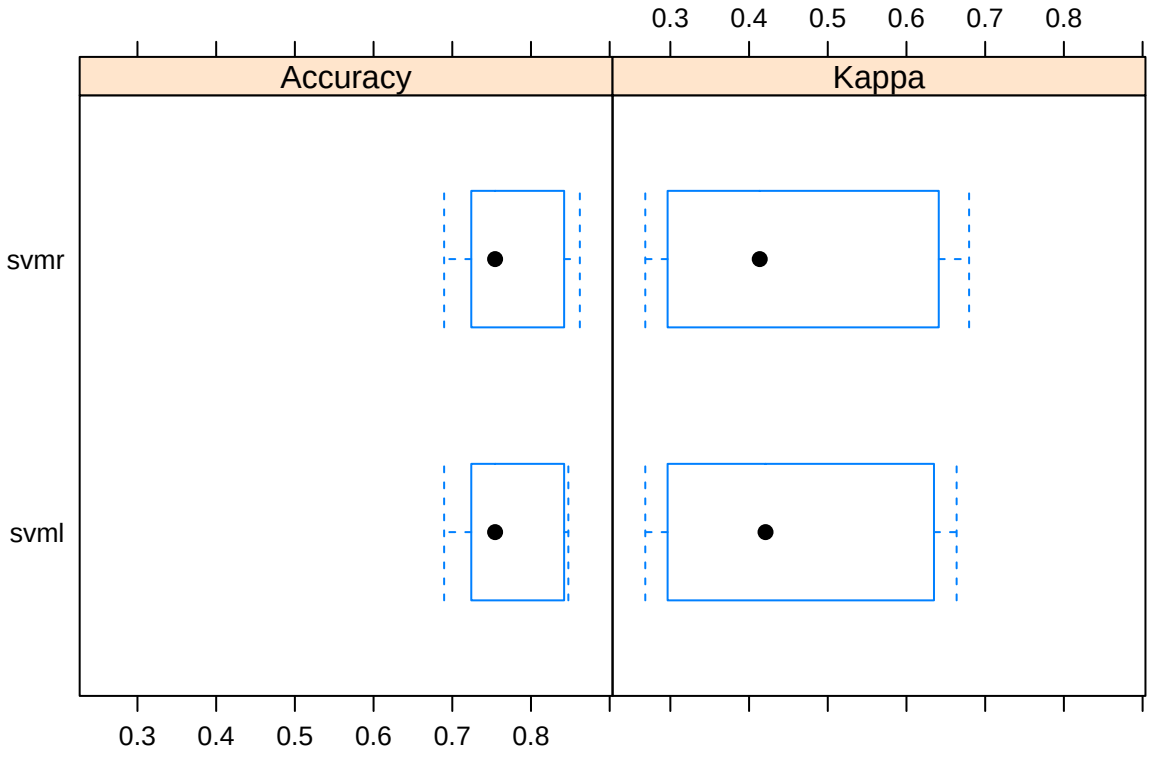
```
ctrl <- trainControl(method = "cv")

set.seed(1)
svml.fit <- train(diabetes~.,
                  data = dat[rowTrain,],
                  method = "svmLinear2", # This uses the svm function in e1071 package.
                  # you can specify svmlinear and then change cost to C. This is from a different packa
                  preProcess = c("center", "scale"), # This is done by default in svm function.
                  tuneGrid = data.frame(cost = exp(seq(-5,1,len=20))),
                  trControl = ctrl)

# In SVM we don't get probabilities so we just use accuracy.

ggplot(svml.fit, highlight = TRUE)
```




```
svmr.grid <- expand.grid(C = exp(seq(-4,5,len=10)),
                         sigma = exp(seq(-8,-3,len=5))) # This sigma is the same as gamma in the svm fu
set.seed(1)
svmr.fit <- train(diabetes~., dat,
                  subset = rowTrain,
                  method = "svmRadial",
                  preProcess = c("center", "scale"),
                  tuneGrid = svmr.grid,
                  trControl = ctrl)

ggplot(svmr.fit, highlight = TRUE)
```




```
resamp <- resamples(list(svmr = svmr.fit, svml = svml.fit))
bwplot(resamp)
```



Test data performance

We finally look at the test data performance.

```
pred.svml <- predict(svml.fit, newdata = dat[-rowTrain,])
pred.svmr <- predict(svmr.fit, newdata = dat[-rowTrain,])

confusionMatrix(data = pred.svml,
                reference = dat$diabetes[-rowTrain])
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction pos neg
##        pos  38  16
##        neg  29 109
##
##                Accuracy : 0.7656
##                  95% CI : (0.6992, 0.8236)
##     No Information Rate : 0.651
##     P-Value [Acc > NIR] : 0.0004037
##
##                   Kappa : 0.4599
##  Mcnemar's Test P-Value : 0.0736383
##
##             Sensitivity : 0.5672
##             Specificity : 0.8720
##          Pos Pred Value : 0.7037
##          Neg Pred Value : 0.7899
##              Prevalence : 0.3490
##          Detection Rate : 0.1979
##    Detection Prevalence : 0.2812
##       Balanced Accuracy : 0.7196
##
##        'Positive' Class : pos
##
```

```
confusionMatrix(data = pred.svmr,
                reference = dat$diabetes[-rowTrain])
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction pos neg
##        pos  38  16
##        neg  29 109
##
##                Accuracy : 0.7656
##                  95% CI : (0.6992, 0.8236)
##     No Information Rate : 0.651
##     P-Value [Acc > NIR] : 0.0004037
##
##                   Kappa : 0.4599
##  Mcnemar's Test P-Value : 0.0736383
##
```

```
##             Sensitivity : 0.5672
##             Specificity : 0.8720
##          Pos Pred Value : 0.7037
##          Neg Pred Value : 0.7899
##              Prevalence : 0.3490
##          Detection Rate : 0.1979
##    Detection Prevalence : 0.2812
##       Balanced Accuracy : 0.7196
##
##        'Positive' Class : pos
##
```