

Practical session: Introduction to SVM in R

Jean-Philippe Vert

November 23, 2015

In this session you will

- Learn how to manipulate a SVM in R with the package kernlab
- Observe the effect of changing the C parameter and the kernel
- Test a SVM classifier for cancer diagnosis from gene expression data

1 Linear SVM

Here we generate a toy dataset in 2D, and learn how to train and test a SVM.

1.1 Generate toy data

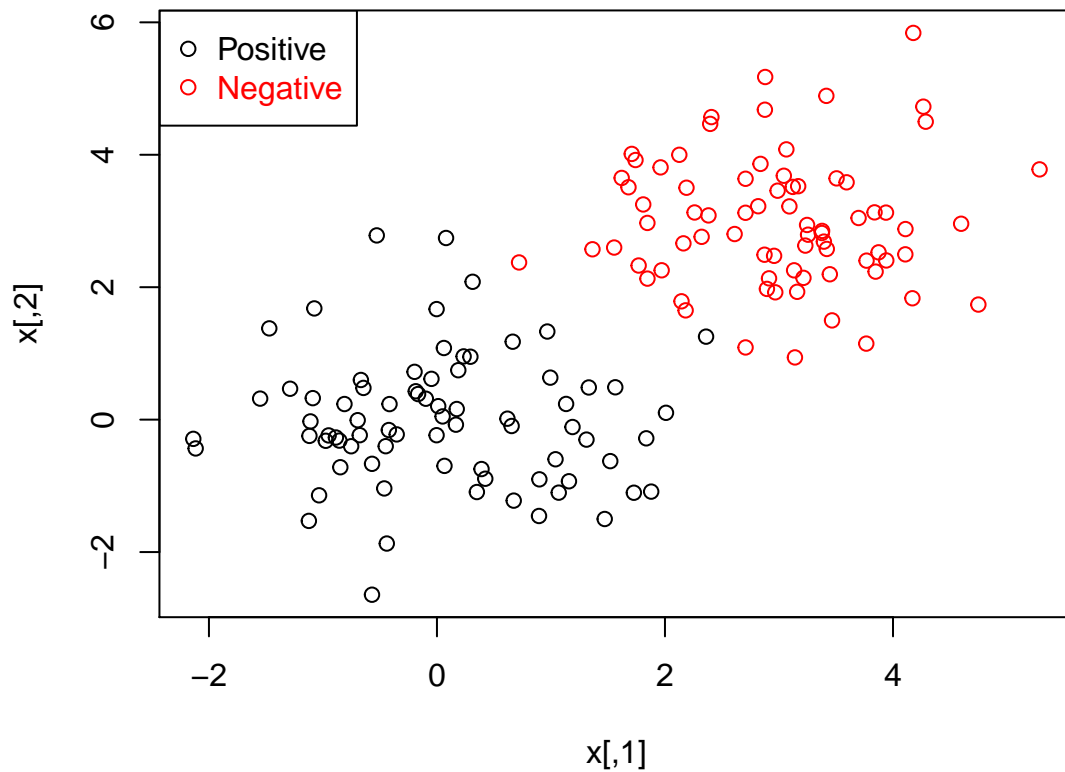
First generate a set of positive and negative examples from 2 Gaussians.

```
n <- 150 #number of data points
p <- 2 # dimension
sigma <- 1 # variance of the distribution
meanpos <- 0 # centre of the distribution of positive examples
meanneg <- 3 # centre of the distribution of negative examples
npos <- round(n / 2) # number of positive examples
nneg <- n - npos # number of negative examples

# Generate the positive and negative examples
xpos <- matrix(rnorm(npos * p, mean = meanpos, sd = sigma), npos, p)
xneg <- matrix(rnorm(nneg * p, mean = meanneg, sd = sigma), nneg, p)
x <- rbind(xpos, xneg)

# Generate the labels
y <- matrix(c(rep(1, npos), rep(-1, nneg)))

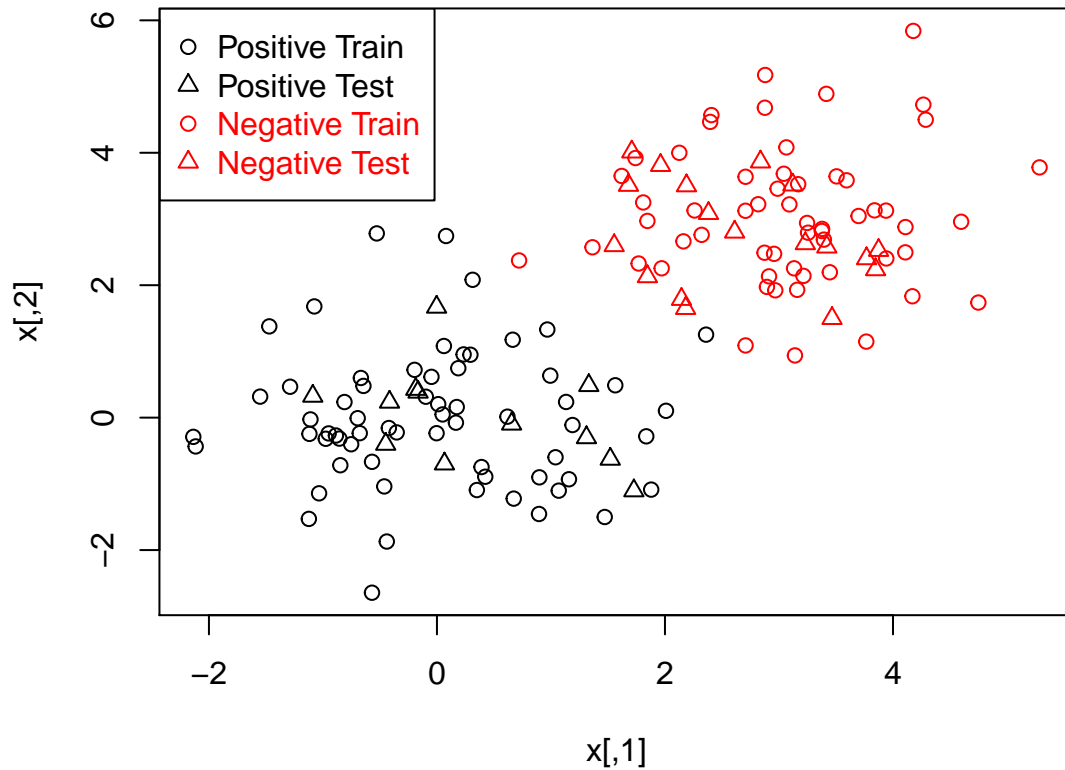
# Visualize the data
plot(x, col = ifelse(y > 0, 1, 2))
legend("topleft", c("Positive", "Negative"), col = seq(2), pch = 1, text.col = seq(2))
```



Now we split the data into a training set (80%) and a test set (20%)

```
# Prepare a training and a test set
ntrain <- round(n * 0.8) # number of training examples
tindex <- sample(n, ntrain) # indices of training samples
xtrain <- x[tindex, ]
xtest <- x[-tindex, ]
ytrain <- y[tindex]
ytest <- y[-tindex]
istrain <- rep(0, n)
istrain[tindex] <- 1

# Visualize
plot(x, col = ifelse(y > 0, 1, 2), pch = ifelse(istrain == 1, 1, 2))
legend("topleft", c("Positive Train", "Positive Test", "Negative Train", "Negative Test"), col = c(1, 1, 2, 2))
```



1.2 Train a SVM

Now we train a linear SVM with parameter $C=100$ on the training set.

```
# load the kernlab package
# install.packages("kernlab")
library(kernlab)

# train the SVM
svp <- ksvm(xtrain, ytrain, type = "C-svc", kernel = "vanilladot", C=100, scaled=c())

# Look and understand what svp contains
# General summary
svp

# Attributes that you can access
attributes(svp)

# For example, the support vectors
alpha(svp)
alphaindex(svp)
b(svp)
```

```
# Use the built-in function to pretty-plot the classifier
plot(svp, data = xtrain)
```

QUESTION1 - Write a function `plotlinearsvm=function(svp,xtrain)` to plot the points and the decision boundaries of a linear SVM, as in Figure 1. To add a straight line to a plot, you may use the function `abline`.

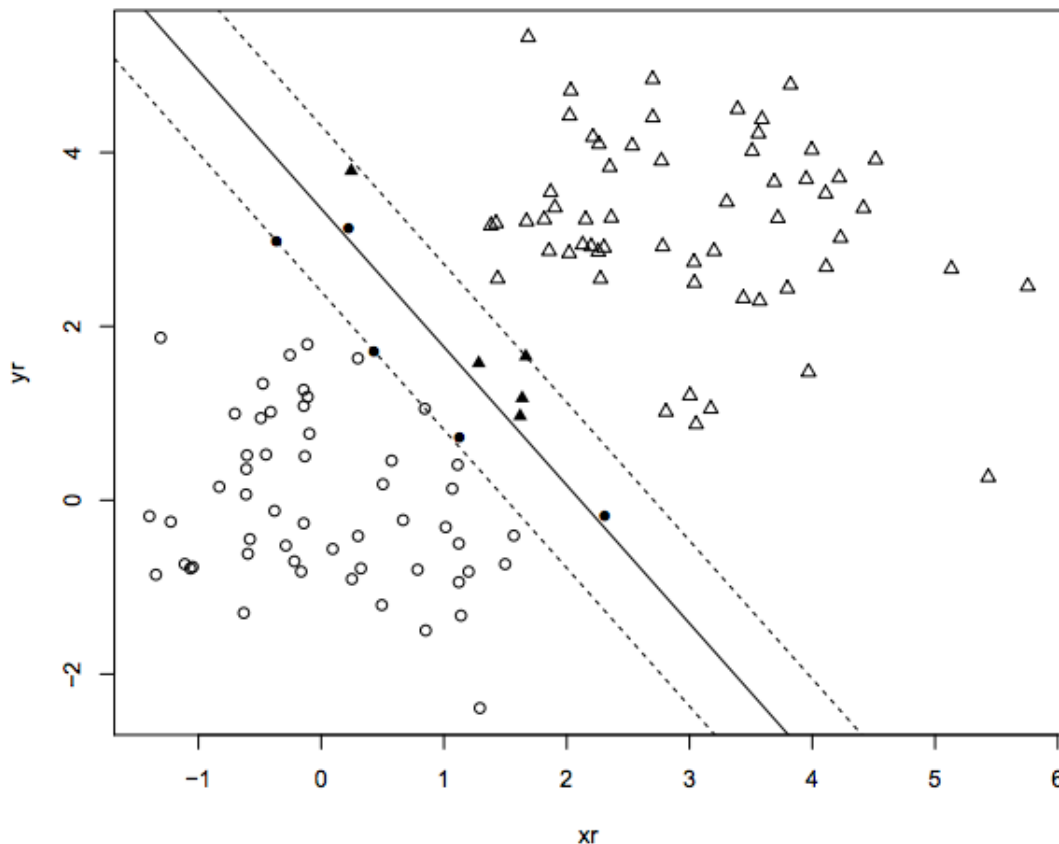


Figure 1: A linear SVM with decision boundary $f(x) = 0$. Dotted lines correspond to the level sets $f(x) = 1$ and $f(x) = -1$ Support vectors are in black.

1.3 Predict with a SVM

Now we can use the trained SVM to predict the label of points in the test set, and we analyze the results using variant metrics.

```
# Predict labels on test
ypred <- predict(svp, xtest)
table(ytest, ypred)
```

```

# Compute accuracy
sum(ypred == ytest) / length(ytest)

# Compute at the prediction scores
ypredscore <- predict(svp, xtest, type = "decision")

# Check that the predicted labels are the signs of the scores
table(ypredscore > 0, ypred)

# Package to compute ROC curve, precision-recall etc...
# install.packages("ROCR")
library(ROCR)

## Loading required package: gplots
##
## Attaching package: 'gplots'
##
## The following object is masked from 'package:stats':
##
##   lowess

pred <- prediction(ypredscore, ytest)

# Plot ROC curve
perf <- performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf)
# Plot precision/recall curve
perf <- performance(pred, measure = "prec", x.measure = "rec")
plot(perf)
# Plot accuracy as function of threshold
perf <- performance(pred, measure = "acc")
plot(perf)

```

1.4 Cross-validation

Instead of fixing a training set and a test set, we can improve the quality of these estimates by running k-fold cross-validation. We split the training set in k groups of approximately the same size, then iteratively train a SVM using k - 1 groups and make prediction on the group which was left aside. When k is equal to the number of training points, we talk of leave-one-out (LOO) cross-validation. To generate a random split of n points in k folds, we can for example create the following function:

```

cv.folds <- function(y, folds = 3){
  ## randomly split the n samples into folds
  split(sample(length(y)), rep(1:folds, length = length(y)))
}

```

QUESTION2 - Write a function `cv.ksvm = function(x, y, folds = 3,...)` which returns a vector `ypred` of predicted decision score for all points by k-fold cross-validation

QUESTION3 - Compute the various performance of the SVM by 5-fold cross-validation. Alternatively, the `ksvm` function can automatically compute the k-fold cross-validation accuracy:

```
svp <- ksvm(x, y, type = "C-svc", kernel = "vanilladot", C = 100, scaled=c(), cross = 5)
print(cross(svp))
print(error(svp))
```

QUESTION4 - Compare the 5-fold CV estimated by your function and ksvm.

1.5 Effect of C

The C parameters balances the trade-off between having a large margin and separating the positive and unlabeled on the training set. It is important to choose it well to have good generalization.

QUESTION5 - Plot the decision functions of SVM trained on the toy examples for different values of C in the range $2^{seq(-10,14)}$. To look at the different plots you can use the function `par(ask=T)` that will ask you to press a key between successive plots. Alternatively, you can use `par(mfrow = c(5,5))` to see all the plots in the same window

QUESTION6 - Plot the 5-fold cross-validation error as a function of C.

QUESTION7 - Do the same on data with more overlap between the two classes, e.g., re-generate toy data with `meaneg` being 1.

2 Nonlinear SVM

Sometimes linear SVM are not enough. For example, generate a toy dataset where positive and negative examples are mixture of two Gaussians which are not linearly separable.

QUESTION8 - Make a toy example that looks like Figure 2, and test a linear SVM with different values of C .

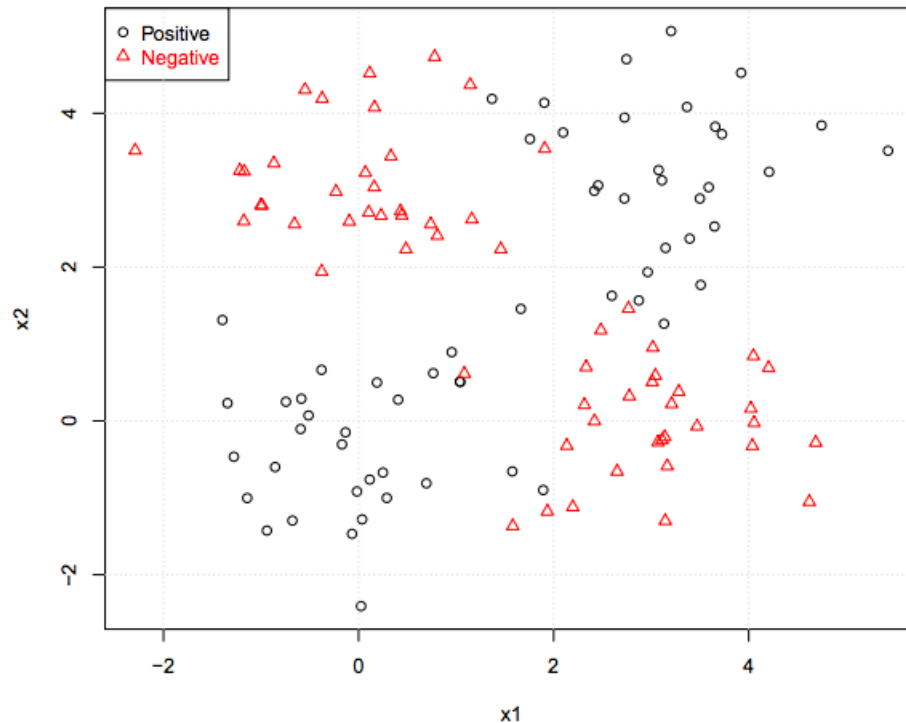


Figure 2: A toy examples where linear SVM will fail.

To solve this problem, we should instead use a nonlinear SVM. This is obtained by simply changing the kernel parameter. For example, to use a Gaussian RBF kernel with $\sigma = 1$ and $C = 1$:

```
# Train a nonlinear SVM
svp <- ksvm(x, y, type = "C-svc", kernel="rbf", kpar = list(sigma = 1), C = 1)

# Visualize it
plot(svp, data = x)
```

You should obtain something that look like Figure 3. Much better than the linear SVM, no? The nonlinear SVM has now two parameters: σ and C . Both play a role in the generalization capacity of the SVM.

QUESTION9 - Visualize and compute the 5-fold cross-validation error for different values of C and σ . Observe their influence.

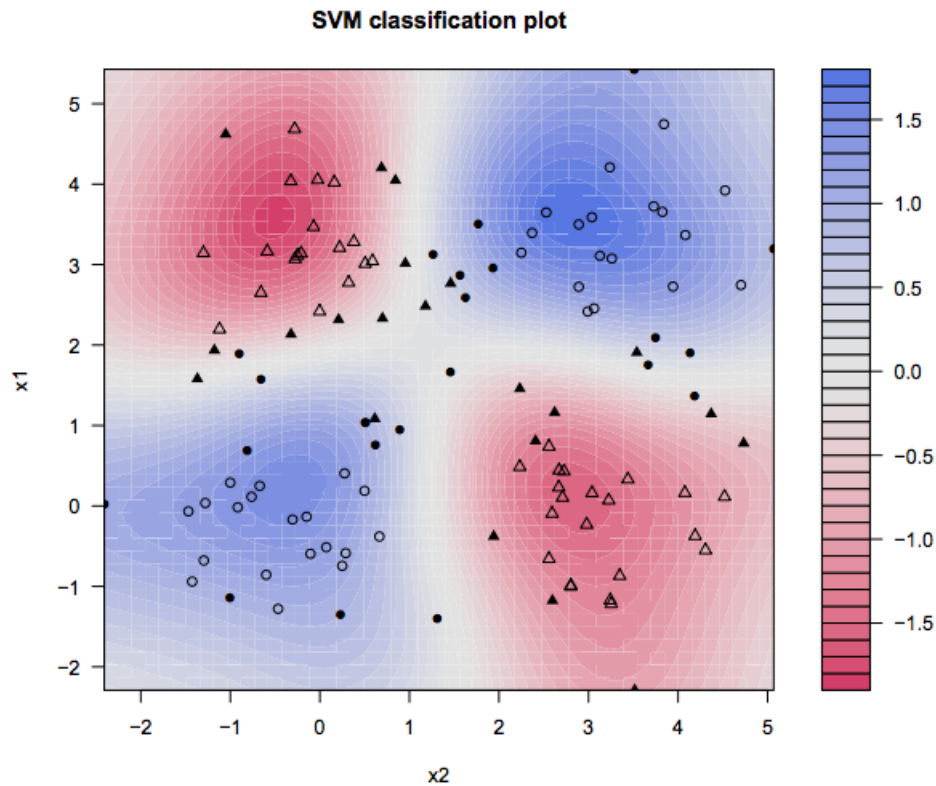


Figure 3: A nonlinear SVM with Gaussian RBF kernel.

A useful heuristic to choose σ is implemented in kernlab. It is based on the quantiles of the distances between the training point.

```
# Train a nonlinear SVM with automatic selection of sigma by heuristic
svp <- ksvm(x, y, type = "C-svc", kernel = "rbf", C = 1)

# Visualize it
plot(svp, data = x)
```

QUESTION10 - Train a nonlinear SVM with various of C with automatic determination of σ . In fact, many other nonlinear kernels are implemented. Check the documentation of kernlab to see them: `?kernels`

QUESTION11 - Test the polynomial, hyperbolic tangent, Laplacian, Bessel and ANOVA kernels on the toy examples.

3 Application: cancer diagnosis from gene expression data

As a real-world application, let us test the ability of SVM to predict the class of a tumour from gene expression data. We use a publicly available dataset of gene expression data for 128 different individuals with acute lymphoblastic leukemia (ALL).

```
# Load the ALL dataset
library(ALL)

## Loading required package: Biobase
## Loading required package: BiocGenerics
## Loading required package: parallel
##
## Attaching package: 'BiocGenerics'
##
## The following objects are masked from 'package:parallel':
##
##   clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##   clusterExport, clusterMap, parApply, parCapply, parLapply,
##   parLapplyLB, parRapply, parSapply, parSapplyLB
##
## The following object is masked from 'package:stats':
##
##   xtabs
##
## The following objects are masked from 'package:base':
##
##   anyDuplicated, append, as.data.frame, as.vector, cbind,
##   colnames, do.call, duplicated, eval, evalq, Filter, Find, get,
##   intersect, is.unsorted, lapply, Map, mapply, match, mget,
##   order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##   rbind, Reduce, rep.int, rownames, sapply, setdiff, sort,
##   table, tapply, union, unique, unlist, unsplit
##
## Welcome to Bioconductor
##
##   Vignettes contain introductory material; view with
##   'browseVignettes()'. To cite Bioconductor, see
##   'citation("Biobase)", and for packages 'citation("pkgname)".

data(ALL)

# Inspect them
?ALL
show(ALL)
print(summary(pData(ALL)))
```

Here we focus on predicting the type of the disease (B-cell or T-cell). We get the expression data and disease type as follows

```
x <- t(exprs(ALL))
y <- substr(ALL$BT, 1, 1)
```

QUESTION12 - Test the ability of a SVM to predict the class of the disease from gene expression. Check the influence of the parameters.

Finally, we may want to predict the type and stage of the diseases. We are then confronted with a multi-class classification problem, since the variable to predict can take more than two values:

```
y <- ALL$BT
print(y)

## [1] B2 B2 B4 B1 B2 B1 B1 B1 B2 B2 B3 B3 B3 B2 B3 B B2 B3 B2 B3 B2 B2 B2
## [24] B1 B1 B2 B1 B2 B1 B2 B B B2 B2 B2 B1 B2 B2 B2 B2 B2 B4 B4 B2 B2 B2
## [47] B4 B2 B1 B2 B2 B3 B4 B3 B3 B3 B4 B3 B3 B1 B1 B1 B1 B3 B3 B3 B3 B3 B3
## [70] B3 B3 B1 B3 B1 B4 B2 B2 B1 B3 B4 B4 B2 B2 B3 B4 B4 B4 B1 B2 B2 B2 B1
## [93] B2 B B T T3 T2 T2 T3 T2 T T4 T2 T3 T3 T T2 T3 T2 T2 T2 T1 T4 T
## [116] T2 T3 T2 T2 T2 T2 T3 T3 T3 T2 T3 T2 T
## Levels: B B1 B2 B3 B4 T T1 T2 T3 T4
```

Fortunately, kernlab implements automatically multi-class SVM by an all-versus-all strategy to combine several binary SVM.

QUESTION13 - Test the ability of a SVM to predict the class and the stage of the disease from gene expression.