1 Introduction

SVM (Support Vector Machine) for classification with R and Python. Study of support points and decision boundaries. Considerations about the determination of the "best" values of the parameters.

This tutorial completes the course material devoted to the Support Vector Machine approach [SVM][1]. It highlights two important dimensions of the method: the position of the support points and the definition of the decision boundaries in the representation space when we construct a linear separator; the difficulty to determine the "best" values of the parameters for a given problem.

We will use R ("e1071" package) and Python ("scikit-learn" package). We focus on didactic aspects in this tutorial. For readers interested in the operational aspects of SVM (learning-test scheme for the evaluation of classifiers, identification of optimal parameters using grid search), I recommend reading our reference document [SVM, section 9]. I recommend also the reading of the tutorials devoted to the studying of the comparison of tools[2] and the behavior of the linear classifiers[3].

2 Support points (vectors) and linear boundaries

2.1 Reminder about SVM

In a binary classification problem, where x is a vector of p descriptors, and y is the class attribute ($y \in \{+1, -1\}$), the SVM approach seeks to construct a linear classification function, at least as a first step:

$$f(x) = x^T\beta + \beta_0$$

The classification rule is: IF $f(x) \geq 0$ THEN $y = +1$ ELSE $y = -1$

The method is based on the maximization of the margin. Two approaches are possible. The primal formulation tries to optimize (when the hard margin with a perfect separation exists) [SVM, page 8]:

[1] [SVM] Tanagra Tutorials, "Support Vector Machine", May 2017.
[2] Tanagra Tutorials, "Implementing SVM on large dataset", July 2009.
[3] Tanagra Tutorials, "Linear classifiers", August 2017.

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2$$
$$s.c.\ y_i \times f(x_i) \geq 1, \forall i = 1,\dots,n$$

$\beta = (\beta_1,\dots,\beta_p)$ is a p-dimensional vector. The aim of the learning process is to estimate the values β and β_0 from a dataset of size n.

The dual formulation (dual problem) emphasizes the notion of support vectors (support points) [SVM, page 13]:

$$\max_{\alpha} L_D(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{i'=1}^{n}\alpha_i\alpha_{i'}y_iy_{i'}\langle x_i, x_{i'}\rangle$$
$$s.c.$$
$$\alpha_i \geq 0, \forall i$$
$$\sum_{i=1}^{n}\alpha_i y_i = 0$$

α_i (i= 1, …, n) is a weight associated to each instance (point). The support points correspond to the instances for which the weight is ($\alpha_i > 0$). In this formulation, the aim of the learning process is to estimate the values of (α_i).

There is a direct relationship between the vectors β and α [SVM, page 15].

2.2 Dataset

The idea is to reproduce the calculations presented in our reference document [SVM, pages 6 to 16] where we calculated the coefficients of the linear boundary and the “margin lines” that perfectly separate the classes in a two-dimensional representation space. I used Excel to explain the process. Let us see if we find the same results using R or Python[4].

We process a dataset with n = 10 instances, p = 2 descriptors $\boldsymbol{x = (x_1, x_2)}$ and a target attribute **y**. We show below the contents of the data file “**data_svm.txt**”.

```
i  x1  x2    y
6  5   1     p
7  7   1     p
8  9   4     p
9  12  7     p
```

[4] A somewhat similar approach is available on the web. Compared to us, the labels are slightly noisy: https://lagunita.stanford.edu/c4x/HumanitiesScience/StatLearning/asset/ch9.html

```
10 13  6      p
1  1   3      n
2  2   1      n
3  4   5      n
4  6   9      n
5  8   7      n
```

Column i is an identifier that is not used in the calculations. It will be used to identify individuals in graphical representations. The target attribute is coded y ∈ {p, n} for convenience. The positive instances are placed before the negative ones into the data file. These has no influence on the calculations.

We use R first. We will reproduce the calculations in Python. This allows us to compare the behavior of these tools in the SVM learning process.

2.3 Analysis in R

2.3.1 Data importation and graphical representations

We read the **“data_svm.txt”** data file with the **read.table()** command. The first row corresponds the name of the variables (header). The first column corresponds to the identifier of the instances (row.names).

```
#loading the data file
df <- read.table("data_svm.txt",header=T,sep="\t",row.names=1)

#descriptive statistics
print(summary(df))
```

Variables and their types are correctly recognized. Column “i” is not used for the calculations.

```
       x1               x2         y    
 Min.   : 1.00   Min.   :1.00   n:5  
 1st Qu.: 4.25   1st Qu.:1.50   p:5  
 Median : 6.50   Median :4.50        
 Mean   : 6.70   Mean   :4.40        
 3rd Qu.: 8.75   3rd Qu.:6.75        
```

We project the points in a two-dimensional representation space by colouring them according to their class membership and by displaying their identifiers (**Erreur ! Source du renvoi introuvable.**).

```
#scatterplot (x1, x2)
plot(df$x1,df$x2,type="n")
text(df$x1,df$x2,rownames(df),col=c("blue","red")[df$y],cex=0.75)
```

Drawing a straight line to separate the red and blue dots without error is possible. We are talking about a separator hyperplane in a space with more than 2 dimensions.

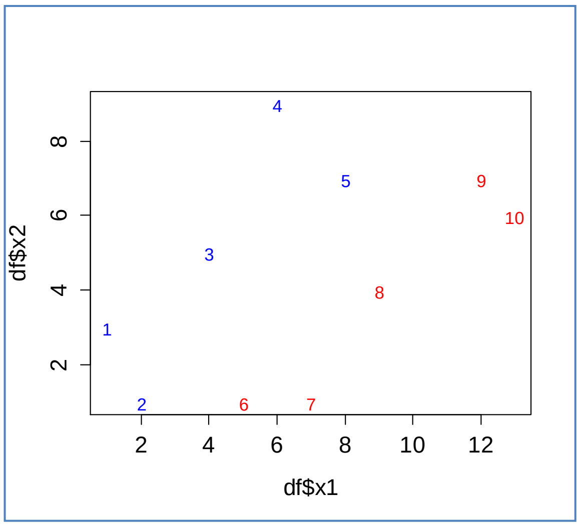

Figure 1 – Scatterplot (x1, x2) - R

2.3.2 Learning process and reading the results

We use the e1071 package for SVM. We call the **svm()** procedure to build a linear classifier (kernel ='linear') and we do not standardize (center and reduce) the variables (scale = F).

```
#loading the package e1071
library(e1071)
#Linear SVM (kernel), the variables are not standardized (scale)
mlin <- svm(y ~ x1+x2, data=df, kernel="linear",scale=F)
print(mlin)
```

The outputs give the number of support points s.

```
Call:
svm(formula = y ~ x1 + x2, data = df, kernel = "linear", scale = F)
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5
Number of Support Vectors:  3
```

Usually, the number of support points is a good indicator. If it is too high in relation to the sample size n, we can legitimately think that modelling is not very effective. Nevertheless, our sample size $n = 10$ is too small for us to really draw conclusions from the value $s = 3$.

The object can give more complete information. We get the list of properties with the command **attributes()** :

```
#properties of the object
print(attributes(mlin))
```

$index notably provides the list of support points.

```
$names
 [1] "call"            "type"           "kernel"         "cost"
 [5] "degree"          "gamma"          "coef0"          "nu"
 [9] "epsilon"         "sparse"         "scaled"         "x.scale"
[13] "y.scale"         "nclasses"       "levels"         "tot.nSV"
[17] "nSV"             "labels"         "SV"             "index"
[21] "rho"             "compprob"       "probA"          "probB"
[25] "sigma"           "coefs"          "na.action"      "fitted"
[29] "decision.values" "terms"

$class
[1] "svm.formula" "svm"
```

These are the instances...

```
#number of the support instances
print((rownames(df))[mlin$index])
```

n° **6**, **2** and **5**.

We highlight them into the scatterplot (Figure 2):

```
#highlighting the support points
points(df$x1[mlin$index],df$x2[mlin$index],cex=3,col=rgb(0,0,0))
```

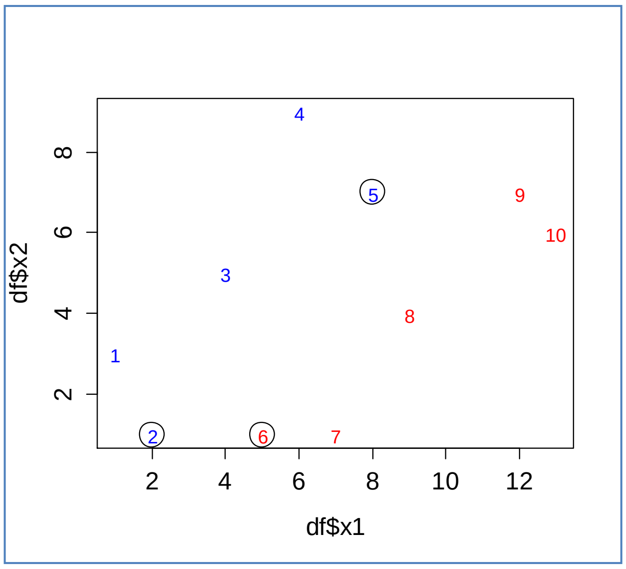

Figure 2 – Highlighting the support points - R

2.3.3 Representation of the decision boundary

Separator hyperplane. To draw the separation line, we must have the coefficients of the equation:

$$\beta_1 x_1 + \beta_2 x_2 + \beta_0 = 0$$

The object *mlin* allows to obtain β_0 with: β_0 = - *mlin$rho*

```
#intercept ➔ -1.666317
beta.0 <- -mlin$rho
print(beta.0)
```

For the other coefficients, we can use the relation [SVM, page 15]:

$$\beta_j = \sum_{i=1}^{n} \alpha_i y_i x_{ij} \text{ ; knowing that } \alpha_i > 0 \text{ only for the support points}$$

The property *mlin$coefs* provides the values of $\alpha_i y_i$ when ($\alpha_i > 0$); to get β_1 and β_2, we must multiply *mlin$coefs* respectively the values of x_1 and x_2

```
#coefficients
beta.1 <- sum(mlin$coefs*df$x1[mlin$index])
beta.2 <- sum(mlin$coefs*df$x2[mlin$index])
print(paste(beta.1,beta.2))
```

We get β_1 = 0.666492 and β_2 = -0.666492

We can draw the straight line using the following equation: $x_2 = -\frac{\beta_1}{\beta_2} x_1 - \frac{\beta_0}{\beta_2}$

```
#drawing the separation line in green
abline(-beta.0/beta.2,-beta.1/beta.2,col="green")
```

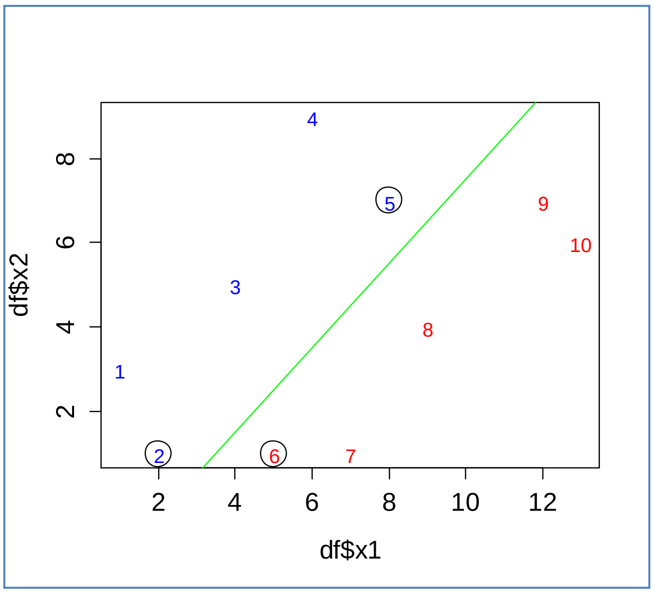

Figure 3- Separation line - R

Observations above the separation line will be classified as negative instances (y = -1), those below will be labelled as positive (y = +1).

Lines delimiting the margins. We represent the "margin" lines which define the support points. They correspond to the saturation of the optimization constraints [SVM, page 8]:

$$y_i \times f(x_i) = 1$$

We have two lines, the first for y_i = -1, the second for y_i = +1

```
#"margin" lines
abline((-beta.0-1.0)/beta.2,-beta.1/beta.2,col="gray")
abline((-beta.0+1.0)/beta.2,-beta.1/beta.2,col="gray")
```

We draw them into the scatterplot,

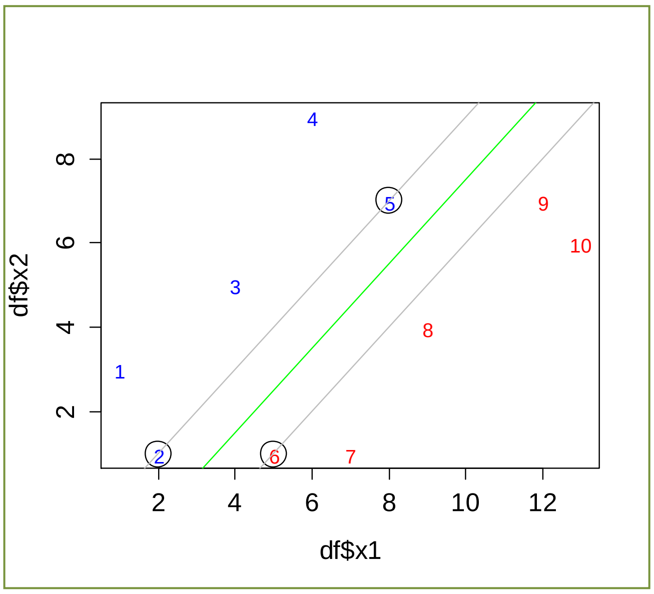

Figure 4 – Graphical representation of the study - R

Note: In the context of perfect separation, the "margin" lines pass exactly over the support points. This is no longer true when we have noisy data i.e. when some points may be located on the wrong side of the decision boundary [SVM, soft margin, pages 20 to 23].

2.4 Analysis in Python

In this section, we reproduce the analysis above in identical form (section 2.3) using the tandem Python / scikit-learn.

2.4.1 A specific function for the graphical representation

Charts appearing several times throughout the study. I have written a specific procedure for organizing the commands needed to represent the point cloud with their identifiers and colors associated with the classes.

```
#function for scatterplot
#input:      data.frame with all the instances
#            data.frame related to positive and negative instances
def myscatter(df,dfpos,dfneg):
    #blank scatterplot for delimiting the chart size
    plt.scatter(df.iloc[:,0],df.iloc[:,1],color="white")

    #annotate - positive instances
    for i in dfpos.index:
        plt.annotate(i,xy=(df.loc[i,'x1'],df.loc[i,'x2']),xytext=(-3,-3),textcoords='offset points',color='red')

    #annotate - negative instances
    for i in dfneg.index:
        plt.annotate(i,xy=(df.loc[i,'x1'],df.loc[i,'x2']),xytext=(-3,-3),textcoords='offset points',color='blue')

    return None
#end of the function
```

plt is an alias for the “matplotlib.pyplot” module.

I use annotate() to display the identifiers into the chart.

Their position had to be shifted slightly [xytext = (-3, -3)] so that they (the identifiers) were positioned exactly at the exact location of the points. Otherwise, the “margin” lines would appear to be shifted compared with the support points.

2.4.2 Data importation and graphical representations

First, we load the dataset and we draw the scatterplot.

```
#package pandas for data handling
import pandas

#load the data file into a Pandas data frame structure
df = pandas.read_table("data_svm.txt",sep="\t",header=0,index_col=0)
print(df.shape)

#split the data into two distinct data.frame: y=+1 (p), y=-1 (n)
dfpos = df[df['y']=='p']
dfneg = df[df['y']=='n']
```

```
#package for plotting
import matplotlib.pyplot as plt

#scatterplot
myscatter(df,dfpos,dfneg)
plt.show()
```

We get…

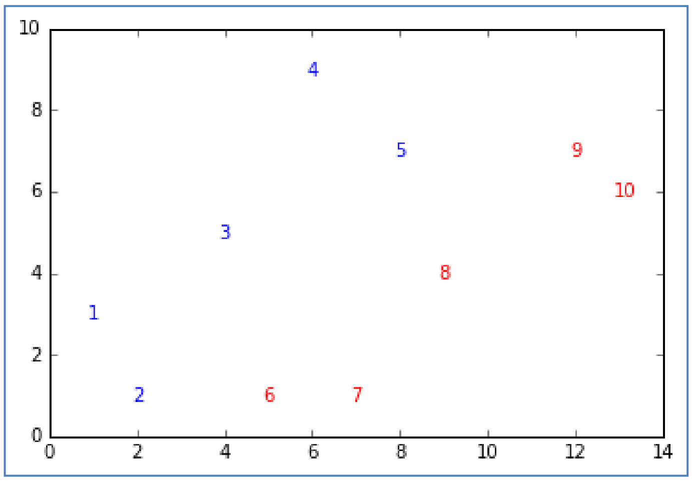

Figure 5 – Scatterplot - Python

… the same scatterplot than in R.

2.4.3 Learning process and reading the results

We use the SVC tool from the scikit-learn package to perform the learning phase. It provides several information:

```
#importing SVC
from sklearn.svm import SVC

#instantiation of the object: linear kernel
svm = SVC(kernel='linear')

#learning process
svm.fit(df.as_matrix()[:,0:2],df.as_matrix()[:,2])

#number of support points…
print(svm.support_.shape)

#... and their labels
print(df.index[svm.support_])
```

The results are consistent with those of R.

```
#number of support points
(3,)
#their labels → n°2, 5 and 6
Int64Index([2, 5, 6], dtype='int64', name='i')
```

The weights α_i of the support points are also available

```
#weigth of the support vectors
print(svm.dual_coef_)
```

That is [SVM, page 14],

```
#αi for i = 2, 5 et 6
[[-0.33325096 -0.11109464  0.44434559]]
```

We draw them into our scatterplot.

```
#coordinates of the support points: c1 for x1, c2 for x2
c1 = svm.support_vectors_[:,0]
c2 = svm.support_vectors_[:,1]

#highlighting the support points
myscatter(df,dfpos,dfneg)
plt.scatter(c1,c2,s=200,facecolors='none',edgecolors='black')
plt.show()
```

The chart corresponds to the Figure 2 for R:

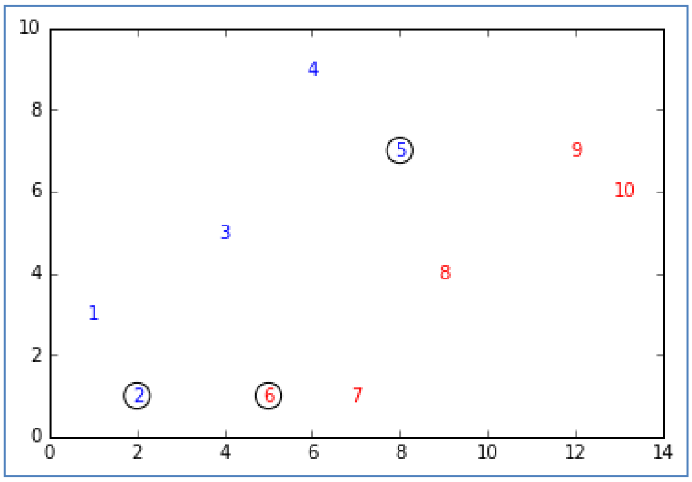

Figure 6 – Highlighting the support points - Python

2.4.4 Representation of the decision boundary

SVC incorporates an interesting functionality. The object can automatically provide the coefficients of the separation line when the kernel is linear.

```
#coefficients β1 and β2
print(svm.coef_)
```

```
#intercept β0
print(svm.intercept_)
```

I did not find the equivalent of the abline() function of R. The trick here is to calculate the coordinates of two points, then draw a line that connects them.

```
#coordinates of the points for representing the separation line and the margins
import numpy as np
xf = np.array([3,12])
yf = -svm.coef_[0][0]/svm.coef_[0][1]*xf-svm.intercept_/svm.coef_[0][1]
xb = np.array([4.5,12])
yb = -svm.coef_[0][0]/svm.coef_[0][1]*xb-(svm.intercept_-1.0)/svm.coef_[0][1]
xh = np.array([2,11])
yh = -svm.coef_[0][0]/svm.coef_[0][1]*xh-(svm.intercept_+1.0)/svm.coef_[0][1]

#graphical representation
myscatter(df,dfpos,dfneg)
plt.scatter(c1,c2,s=200,facecolors='none',edgecolors='black')
plt.plot(xf,yf,c='green')
plt.plot(xb,yb,c='gray')
plt.plot(xh,yh,c='gray')
plt.show()
```

The chart (Figure 7) is equivalent to the one of R (Figure 4):

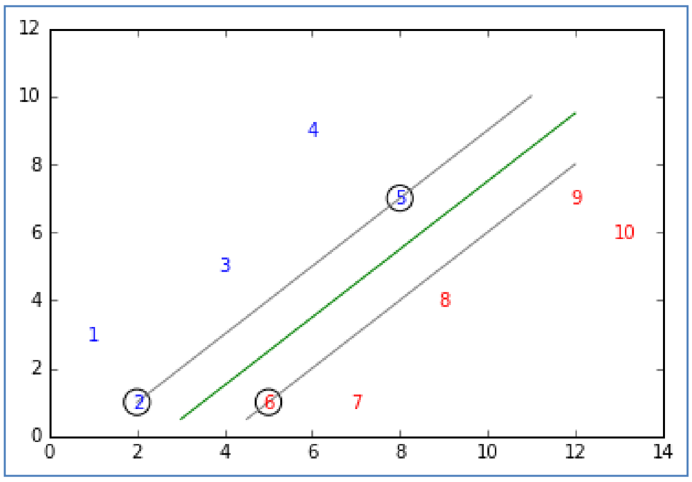

Figure 7- Graphical representation in Python

3 Importance of the parameters

3.1 Nonlinear SVM – Using kernel functions

Margin maximization is a possible approach for dealing with a classification problem, as well as maximizing likelihood for instance. One of the main advantage of the SVM is the utilization of the kernel functions $K(x_i,x_{i'})$ in the dual form. They allow us to project the instances into a larger space without having to explicitly create the intermediate variables. In fact, we can customize the presentation power of our classifier.

In the "soft margin" context, the optimization problem becomes [SVM, page 28]

$$\max_{\alpha} L_D(\alpha)=\sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i=1}^{n}\sum_{i'=1}^{n}\alpha_i\alpha_{i'}y_iy_{i'}K(x_i,x_{i'})$$

$$s.c.$$

$$\sum_{i=1}^{n}\alpha_i y_i=0$$

$$0\le\alpha_i\le C,\ \forall i$$

The "cost parameter" **C** is fundamental. It indicates our tolerance for misclassified instances. If it is too high combined with an inadequate kernel function, it can lead us to overfitting because our model overreacts to the learning sample. If it is too low, we do not sufficiently extract valuable information from the data, leading to underfitting.

Setting the right value of **C** is not obvious. In our study, we deal with an artificial dataset. It will be easy to find the right solution. In real situations, on data and problems for which we do not control all the ins and outs, we must adopt a trial and error approach.

In spite of this, I note that the issue of the kernel function is often raised in scientific publications. In contrast, the issue of C is hardly addressed in experiments, whereas it has a great influence on the classifier performance.

We exclusively conduct our study with R in this part. The reader should be able to easily transpose the operations in Python.

3.2 Goal and dataset

We work in a 2-dimensional representation space to facilitate the understanding the location of points. We randomly generated data in the square [0 ; 1], in which we included an isosceles triangle. Individuals within the triangle belong to the first class, those outside the triangle belong to the second class.

We generate $n = 30$ instances for the training phase, and $n_{test} = 5000$ as test set.

```
#to get the same dataset at each run
set.seed(10)

#function which generates the dataset (n: dataset size)
generate.data <- function(n){
  x2 <- runif(n)
  x1 <- runif(n)
  y <- factor(ifelse((x2>2*x1)|(x2>(2-2*x1)),1,2))
  return(data.frame(x1,x2,y))
}

#training set - n = 30
dtrain <- generate.data(30)

#graphical representation
plot(dtrain$x1,dtrain$x2,col=c("blue","red")[dtrain$y],xlim=c(0,1),ylim=c(0,1))
abline(0,2)
abline(2,-2)

#test set - ntest = 5000
dtest <- generate.data(5000)

#graphical representation
plot(dtest$x1,dtest$x2,col=c("blue","red")[dtest$y],xlim=c(0,1),ylim=c(0,1))
abline(0,2)
abline(2,-2)
```

There are two comments here (Figure 8):

1. Perfect discrimination is possible. But the classification function must be able to represent a triangle. This has an influence on the bias part of the error.
2. The learning sample does not "fill" the representation space sufficiently, especially in the lower part of the triangle (red dots). This will create uncertainty and has an influence on the variance part of the error.

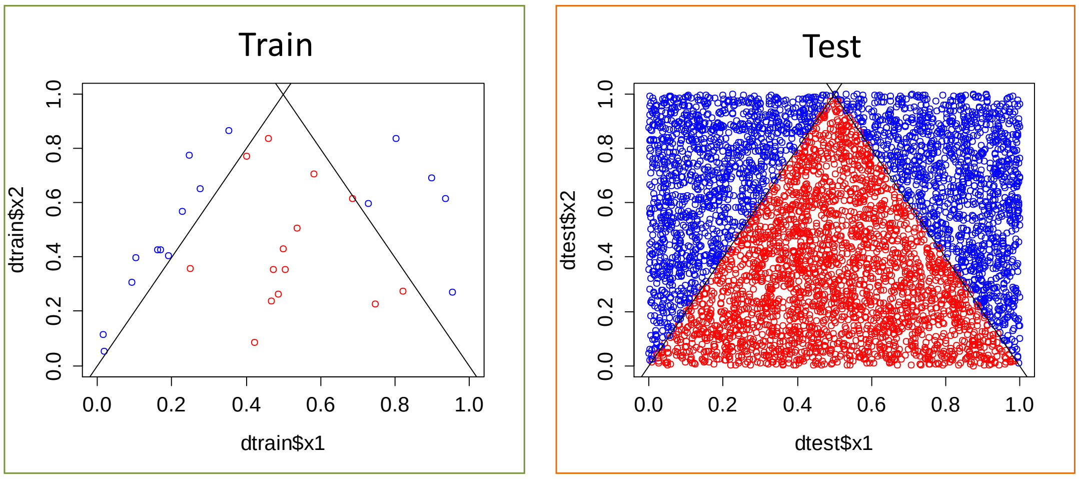

Figure 8 – Triangle problem - Train and test samples

3.3 SVM – Linear kernel

Obviously, a straight line does not separate the blue dots from the red ones. But without a graphical representation, we would not know that. Let us ignore this information, we launch a linear SVM.

```
#e1071
library(e1071)
#Linear SVM
mlin <- svm(y ~ x1+x2, data=dtrain, kernel="linear",scale=F)
print(mlin)
#graphical representation
plot(dtrain$x1,dtrain$x2,col=c("blue","red")[dtrain$y],xlim=c(0,1),ylim=c(0,1))
points(dtrain$x1[mlin$index],dtrain$x2[mlin$index],pch=5,cex=1.75,col=rgb(0,0,0))
abline(0,2)
abline(2,-2)
```

$s = 28$ supports points are found…

```
Call:
svm(formula = y ~ x1 + x2, data = dtrain, kernel = "linear", scale = F)
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5
Number of Support Vectors:  28
```

… almost all points in the learning sample. That is not a very good sign. Into the scatterplot (Figure 9), we note that there is a problem. The support points are scattered everywhere, their positions have absolutely nothing to do with a triangle.

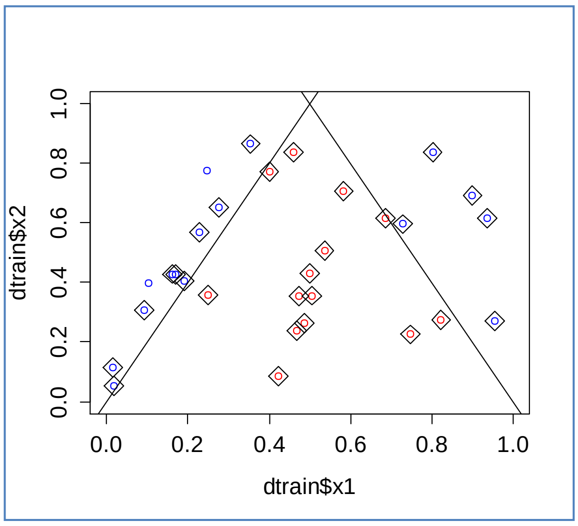

Figure 9 – Supports points supports (diamond) – Linear kernel

If we draw the decision boundary,

```
#intercept β0
beta.0 <- -mlin$rho

#coefficients β1 and β2
beta.1 <- sum(mlin$coefs*dtrain$x1[mlin$index])
beta.2 <- sum(mlin$coefs*dtrain$x2[mlin$index])

#separation line
plot(dtrain$x1,dtrain$x2,col=c("blue","red")[dtrain$y],xlim=c(0,1),ylim=c(0,1))
abline(-beta.0/beta.2,-beta.1/beta.2,col="green")
```

we find it completely inadequate.

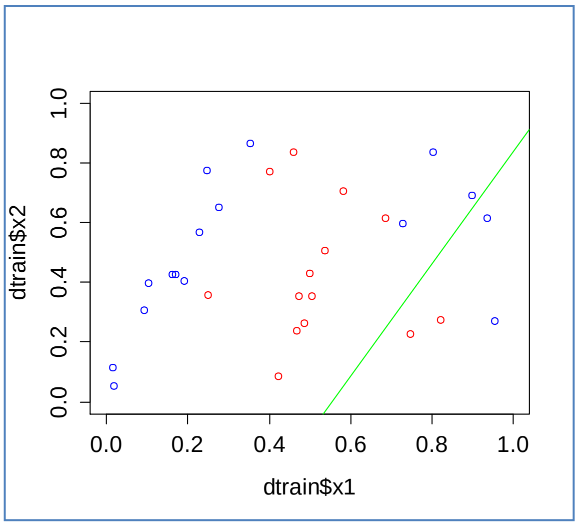

Figure 10 – Separation line – Linear kernel

We can confirm this graphical diagnosis by calculating the confusion matrix on the learning sample:

```
#prediction on the training sample, confusion matrix
ylin.train <- predict(mlin,dtrain)
table(dtrain$y,ylin.train)
```

The result is consistent with what we observe in Figure 10.

```
   ylin.train
     1  2
  1 14  2
  2 12  2
```

Measuring the performance on the test sample is not required here. The result will be equally catastrophic. A linear classifier is clearly not appropriate in our situation.

3.4 SVM – Polynomial kernel

3.4.1 Why a polynomial kernel?

If we consider the organization of our points into the scatterplot (Figure 8), we can think that it is possible to approximate the true decision boundary (in the form of a triangle) with a parabola. The equation would be:

$$x_2 = ax_1^2 + bx_1 + c$$

A kernel polynomial may be appropriate [SVM, page 29]. We set "**coef0 = 1**" so that the terms (x_1, x_1^2,x_2) should be considered in modelling [SVM, page 27].

We launch again the learning process.

```
#using a polynomial kernel of degree 2
mpoly <- svm(y ~ x1+x2, data=dtrain, kernel="polynomial", scale=F, coef0=1, degree=2)
print(mpoly)
```

We do not set the cost parameter. By default, **svm()** uses "**C = 1**". This is important for understanding the results below.

R output announces 29 support points. That is not a very good sign again.

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1
     degree:  2
      gamma:  0.5
     coef.0:  1
Number of Support Vectors:  29
```

We apply the classifier on the test set.

```
#prediction on the test set
ypoly.test <- predict(mpoly,dtest)

#confusion matrix
mc.poly <- table(dtest$y,ypoly.test)
print(mc.poly)

#test error rate
err.poly <- 1-sum(diag(mc.poly))/sum(mc.poly)
print(err.poly)
```

The test error rate is 49.4%. It is very bad.

```
#confusion matrix
   ypoly.test
       1    2
  1 1987  530
  2 1940  543
```

When we visualize the decision boundary into the scatterplot....

```
#plotting
plot(dtest$x1,dtest$x2,col=c("blue","red")[ypoly.test],xlim=c(0,1),ylim=c(0,1))
abline(0,2,lwd=2)
abline(2,-2,lwd=2)
```

… we observe that the classifier draws a straight line to attempt discriminate the classes. We know that this is not the right solution.

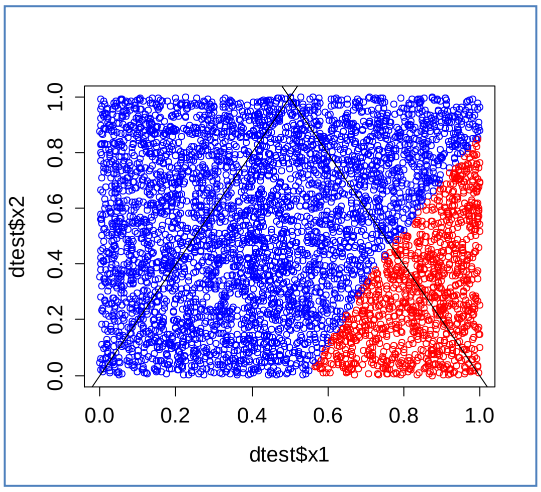

Figure 11 – Decision boundary – SVM – Polynomial kernel of degree 2 (C = 1)

However, the idea of approximating the triangle by a parabola seems logical. Why does the method ignore it and instead draw a straight line?

3.4.2 Modifying the cost parameter

If the parabola seems adequate to approximate the underlying concept, the inadequacies necessarily result from a poor parameterization of the modeling algorithm. In the case of SVM, the cost parameter, which corresponds roughly to a tolerance to the errors, has a crucial influence. On our artificial data, we know that perfect discrimination is possible, we have non-noisy data, and the dimensionality is low (*p = 2* descriptors for *n = 30* observations). We can increase highly the value of **C**. We set "C = 1000".

```
#new setting: C = 1000
mpoly <- svm(y ~ x1+x2, data=dtrain, kernel="polynomial", scale=F, cost=1000, coef0=1, degree=2)
print(mpoly)
```

svm() find 10 support points…

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1000
     degree:  2
      gamma:  0.5
     coef.0:  1
Number of Support Vectors:  10
```

… which are conveniently positioned along the triangle, especially in its upper part.

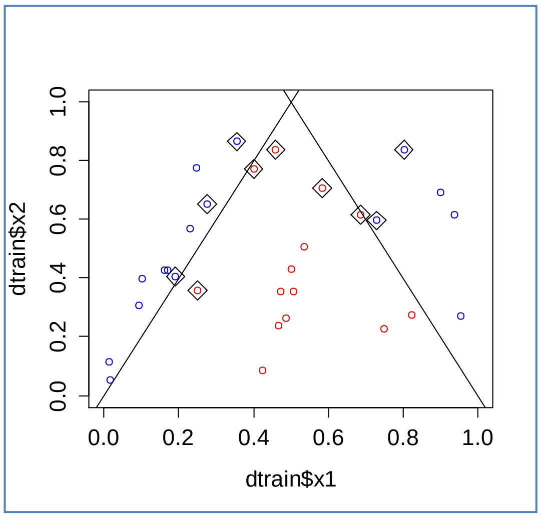

Figure 12 – Support points (diamond) - Polynomial kernel of degree 2 - C = 1000

We can expect better performance and, indeed, the test error rate is 10.76%.

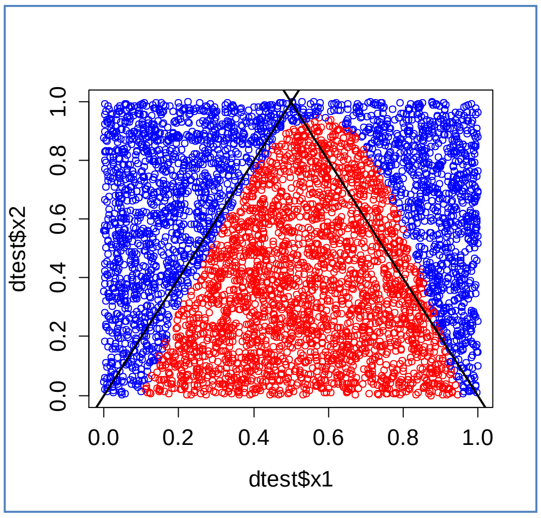

Figure 13 - Frontière induite par un noyau polynomial de degré 2 (C = 1000)

When we draw the decision boundary into the scatterplot, we note that the approximation of the triangle by a parabola was a viable idea. Errors are caused by the small size of the learning sample, which results in the absence of red support points in the lower part of the triangle (Figure 12).

Note 1: The reader can easily verify this by modifying the R program that accompanies this document: when we increase the size of the learning sample, the opposing color support points will line up better along the edges of the triangle, the parabola is better positioned and, consequently, the error rate naturally decreases.

Note 2: If we had left coef0 at its default value (coef0 = 0), x_1 and x_2 do not appear in the transformation underlying the kernel function [SVM, page 27]. It is impossible to obtain a decision boundary taking the form of a parabola, the modeling is bad whatever the value of the cost parameter. We are in a situation similar to that observed for the linear kernel.

Note 3: Not reported here but available in the R program, the RBF kernel (radial basis function) also gains efficiency when we increase the value of the cost parameter C. We reach a test error rate of 6.78%.

4 Conclusion

This tutorial is essentially for pedagogical purposes. In the first part, I tried to illustrate concretely the countless difficult formulas that can be found in the very many course materials that present the SVM approach. I often tell my students that dissecting a mathematical expression is only of interest, at least in our fields, if it allows us to better understand the ideas underlying the method. I had tried to detail the calculations in Excel in the course material prior to this document. The idea was to replicate the approach by using R and Python with the packages dedicated to SVM.

In the second part, we were interested in the parameters. The example allows to comprehend the main difficulties. In addition to the choice of the kernel, the cost parameter C, often ignored in published experimental results, plays a fundamental role. No one sets a value C = 1000 in real problems (in the publications I read in any case), but we realized that it is a value that can be considered in our very specific configuration (absence of noise, low dimensionality). Above all, this means that special attention must be paid to the parameters when we conduct an analysis.

5 References

Abe S., « Support Vector Machines for Pattern Classification », Springer, 2010.

Cristianini N., Shawe-Taylor J., « Support Vector Machines and other kernel-based learning algorithms », Cambridge University Press, 2000.