LAB2
Support Vector Machines

DD2421 – Machine Learning
21st September 2023

FLANDRE Corentin

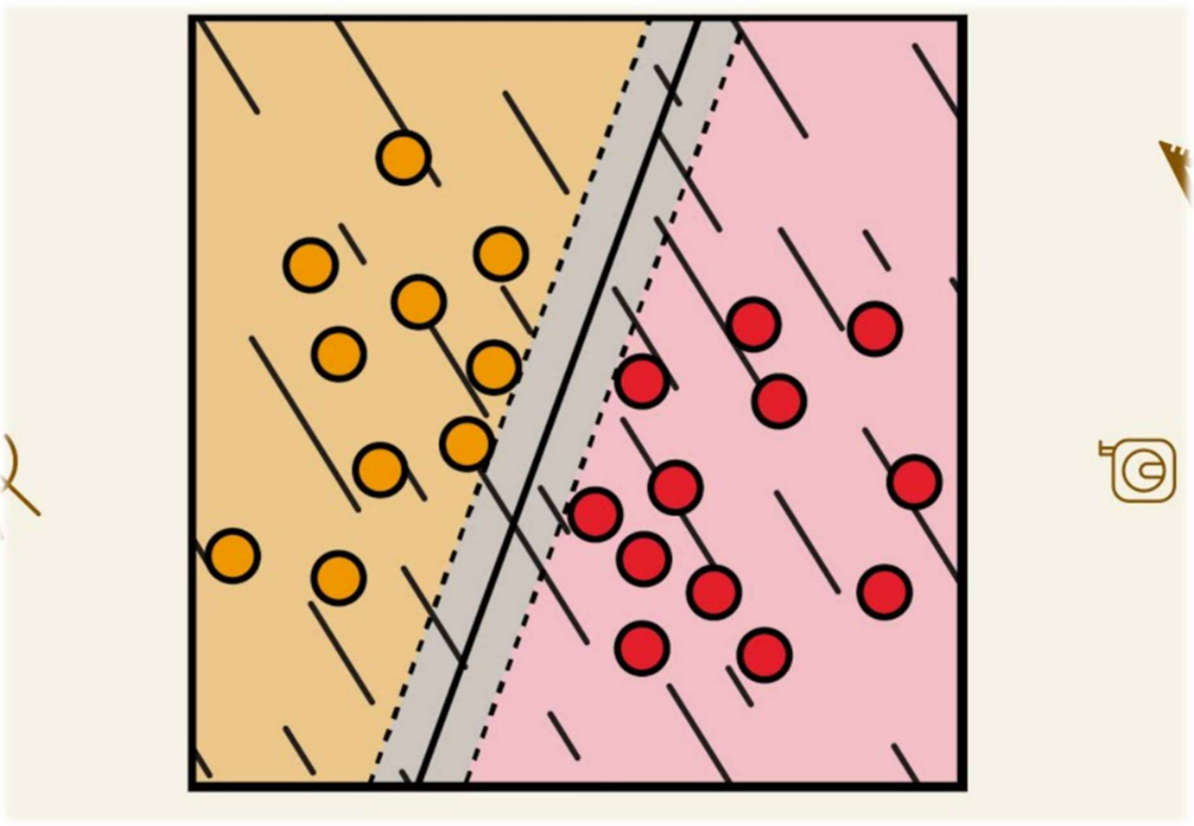


Table of Contents

1. Introduction

The objective of this laboratory is to build a Support Vector Machine (SVM) for classification. It implies to understand and implements optimization problem solver which emerges in the dual formulation of the SVM. During this laboratory, multiples concepts are developed:

- Decision boundary from kernel SVM model
- Dual Formulation Theory
- Support Vector and Margin
- Slack Variables
- Kernel Functions (especially Linear Kernel, Polynomial Kernel, RBF (Radial Basis Functions) Kernel)

During this laboratory, we will increase our skills using packages used in Machine Learning in a classical Python environment such as *numpy*, *scipy* and *matplotlib*. Moreover, the mathematics behind concepts of SVM are important and will give us very good approaches of theoretical machine learning.

The subject suggests that we can explore out more about the SVM thematic paying attention to :
- reaction of the optimize when it's not able to find a solution
- implementation of non-linear kernels and impact on classification
- influence of parameters of non-linear kernels in terms of the bias-variance trade-off
- exploring slack parameter C in large/small values (increase/decrease value)
- balancing between opt for more slack or for more complex model

This is the GitHub repository containing initial files, our codes and the report needed to complete the lab: https://github.com/flandrecorentin/DD2421-ML. The most important files in the repository LAB2-svm are *LAB2-dev.py* corresponding to the code developed to respond in the subject and *kernel_functions.py* corresponding to implementation of used kernel functions.

2. Support Vector Machine

2.1 Theory of SVM

The idea is to build a classifier capable of create a separation (representable under the image of a **decision boundary** to delimit classes. The objective of that model is to search the best delimitation who give the maximal margins to the available data points. Initially an optional transformation of inputs is possible. The location of the decision boundary is given by the weights ($\vec{w}$) and the bias (b), so the problem of SVM is to find these values to maximize the **margin**. Mathematically, this problem is:

$$\min_{\vec{w},b} \overrightarrow{||w||} \quad \text{under the constraint} \quad t_i(\vec{w}^T . \emptyset(\vec{x_i} - b)) \geq 1 \; \forall i$$

Where these different notations mean:

$\vec{w}$	weight vector defining the separating hyperplane
b	offset (bias) for the hyperplane
$\vec{x_i}$	the ith datapoint
t_i	target class (-1 or 1) for data point i
$\emptyset(\dots)$	the optional transformation of input data

With a model with find well adapted value (and the respect of the constraint who assure a certain margin beyond the decision boundary), we can classify a new unknow datapoint using an **indicator** function capable to estimate the class of this new datapoint $\vec{s}$:

$$ind(\vec{s}) = \vec{w}^T . \emptyset(\vec{s}) - b$$

The value returned by the indicator function define the estimate class. If the value is negative, we can conclude that the class is -1 (or the one represents by the -1 value). And vice versa, if the value is positive, we conclude that the sample belongs to the class represented by the value 1.

2.2 Transformation into dual formulation

The problem is transformable to be easily featured with multiples functions (see part 3) more representative of reals classes and with more adaptation against outliers (we can say that's possible to change the trade-off bias-variance of the model). As seen later in the report, opted for the **dual formulation** of the problem allows us to make all these features by changing just a few mathematical points, that's why this dual formulation is used.

One of the most important concepts in SVM classification is **support vectors** (SV). They are linked with datapoints/samples who fit with the decision boundary taking into consideration the margin. These particulars points are on the limit of the decision boundary with margin. The great majority of samples are not concerned by these SV. The indicator values corresponding to these SV have as values 1 or -1 corresponding to the margins lines.

The dual formulation can be represented by the searching of α_i wich minimizes:

$$\frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j t_i t_j K(\vec{x}_i, \vec{x}_j) - \sum_i \alpha_i$$

under the constraints

$$\alpha_i \geq 0 \quad \forall i \text{ and } \quad \sum_i \alpha_i t_i = 0$$

Knowing that $K(\vec{x}_i, \vec{x}_j)$ is called the kernel function seen in the part 3.

Another formula permits to compute the indicator value of an new unknow datapoint using these SV, and another formula to compute the margin (because we know that these SV are on the margin)

2.3 First Implementation of a model

The implementation of a SVM classifier can be divided into multiples steps:

Create the objective function

The *objective* function is used to choose SV vectors, it will modify the alphas values (each alpha corresponding to one vector). This function is called many times, so we must take care of this construction (make it as optimised as possible).

Create the kernel function used

The *kernel function* is used in the objective function (used in the precomputed matrix to accelerate calculation due to the large number of objective calls). It's defined the similarity measure between to point. There are different ways to define this similarity measure as we will see in part 3). The most basic *kernel function* is linear and used the norm 2 scalar product to create linear boundary.

Call minimize function

The hearth of our program to create a SVM classifier model is the *minimize* function of the package *spicy.optimize* We pass the parameters of our specific model : our objective function, our owns constraints

Zerofun function

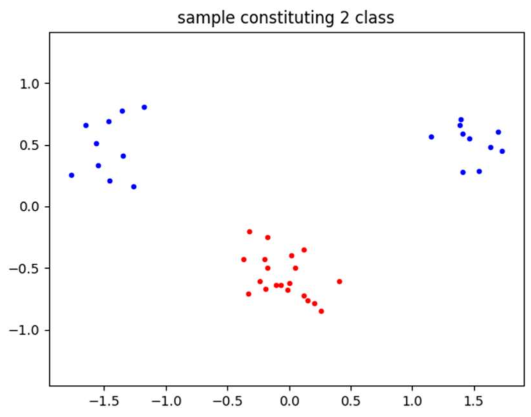
Additionally, to the bounds, some constraints can be imposed. In this case the *zerofun* function permits to constraint a constrained value to 0.

Indicator functions

The *indicator* function aims to estimate the class of a new sample/data. If the value returned by the *indicator* is negative, it's means that the estimated class of the data is the class representing by -1. On the contrary, if the *indicator* is positive, it means that the estimated class is the +1 one. The margins boundaries represent respectively the -1/+1 line indicator.

3. Kernels functions

For this part, the samples used to work on kernels functions is the one proposed by the subject. Indeed, the subject propose to use a 2-dimensional space who is easily representative using plots. Classes/decision boundaries are represented by lines in a 2D-space. With the specific seed (100), we can obtain the same random samples than the subject.



Classes used for testing SVM machines (red and blue)

An important think to know is that some kernels functions use parameters which allow the same function to have different class separations.

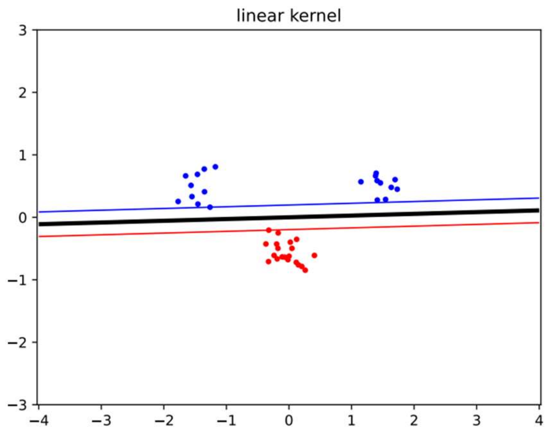
In a next part (part 5), we will try our different SVM models on a more complex sample classification and we will be more available to focus on increasing the success (reduce errors) of the classification model using test samples and other things interesting such as cross validation and etc.

3.1 First results for Linear kernel

The linear kernel is the most basic kernel functions and use the scalar product between two points for the measure:

$$K(\vec{x},\vec{y}) = \vec{x}^T . \vec{y}$$

Basically, the shape of the separation is linear.



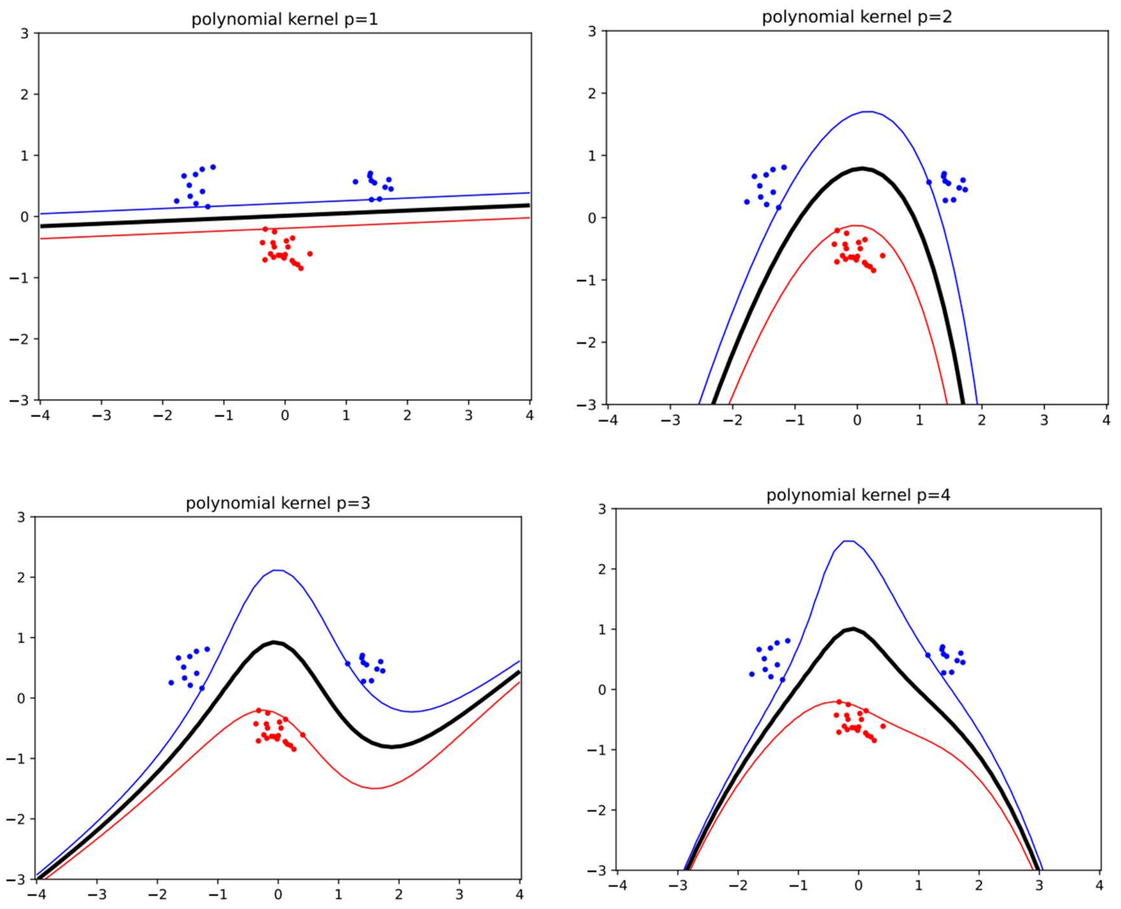
3.2 First results for Polynomial kernel

The polynomial kernel function is a very basic kernel function that's useful when linear classification is no longer relevant.

The polynomial kernel allows curved decision boundaries. This function used a polynomial

$$K(\vec{x},\vec{y}) = (\vec{x}^T . \vec{y} + 1)^p$$

The shapes of the separation line are more complex depending on the value of the parameter p, corresponding to the **dimension of the polynomial** function. With $p = 2$, the shapes of the separation line are quadratics (ellipses, parabolas, hyperbolas...)



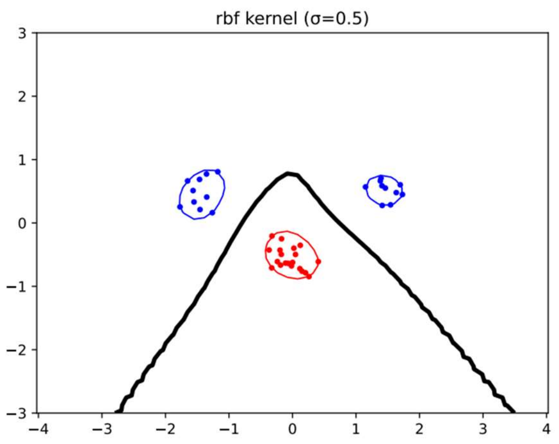
3.3 First results for RBF kernel

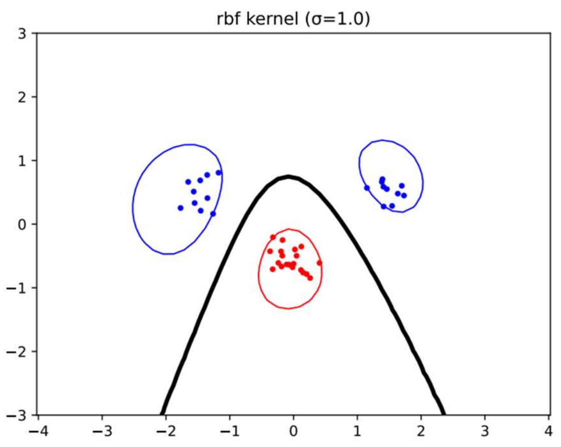
The Radial Basis Functions (RBF) kernel function uses the euclidian distance between two samples and often results in very good boundaries.

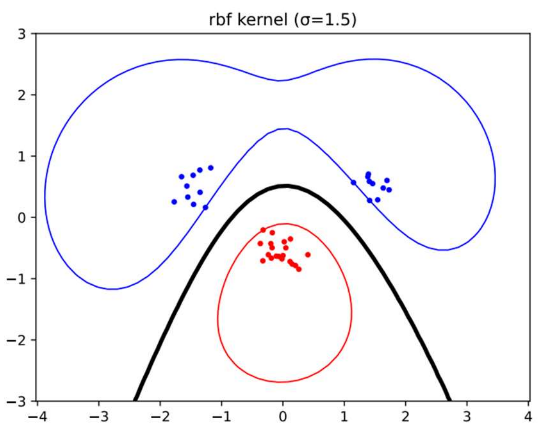
The functions used behind the RBF kernel functions is :

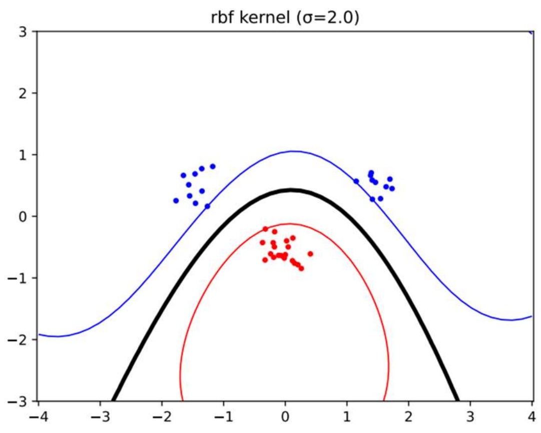
$$K(\vec{x},\vec{y}) = e^{-\frac{||\vec{x}-\vec{y}||^2}{2\sigma^2}}$$

The parameter of the RBF kernel function is used to control the **smoothness of the boundaries** as shown in the following results.


rbf kernel (σ=0.5)


rbf kernel (σ=1.0)


rbf kernel (σ=1.5)


rbf kernel (σ=2.0)

4. Slack Variables and Soft Margins

4.1 Concept of slack variables

To structure the interest of slack variables, one of the interesting examples of the usages of slack variables is maybe a classification which appears globally linear but contains some outliers will make it impossible to classify with hard margins because of these outliers.

I would roughly say that variable slacks are a way of reducing the variance of a model by avoiding the overfitting effect because the model corresponding to a SVM will not necessarily try to divide the classification zones by forcing all the training data to be correctly classified.

Be careful: if the kernel function doesn't allow you to classify correctly, it's not by using variable slacks (implying softs margins) that our classification model will be good. Sometimes it's better to change the kernel function.

The constraint of the SVM problem is more flexible because we suppose that's in our training samples, some sort of **noise** is presented and we allow few datapoints to be miss-classified if it results in wider margin, that's why we call those **soft margins**.

Mathematically, the constraint of the dual formulation of the classification problem change a few:

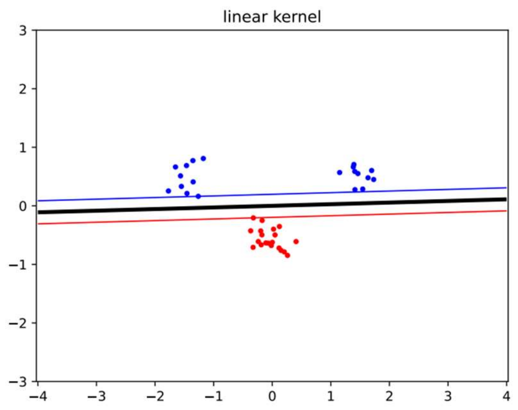
$$C \geq \alpha_i \geq 0 \quad \forall i \text{ and } \quad \sum_i \alpha_i t_i = 0$$

We know that a constraint C is added to our problem who represents the possibility to allow a few errors on the trainings data. C can be seen as a regularization parameter.

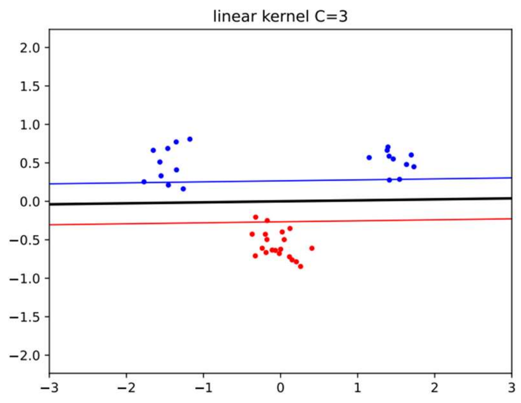
4.2 Usage of slack variables

The important point of concrete usage slack variables is the evolution (increase/decrease) of the C value compared to the impact we want. The lower the value of C, the wider the boundaries will be. Basically, noisy data deserve a low C value.

Example of soft margins with linear kernel function

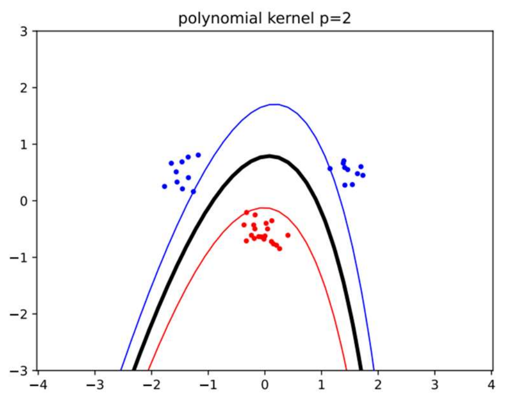


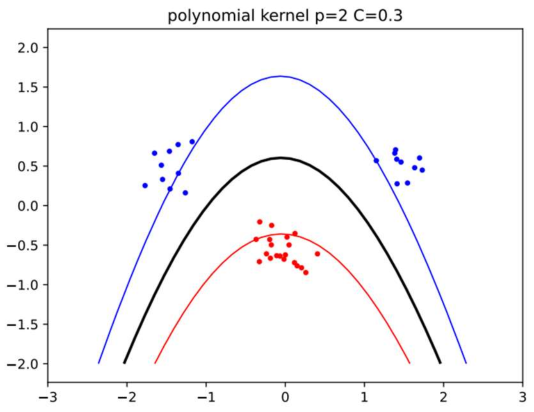
Linear kernel without slack effect



Linear kernel with slack effect

Example of soft margins with polynomial kernel function





Polynomial kernel without slack effect *Polynomial kernel with slack effect*

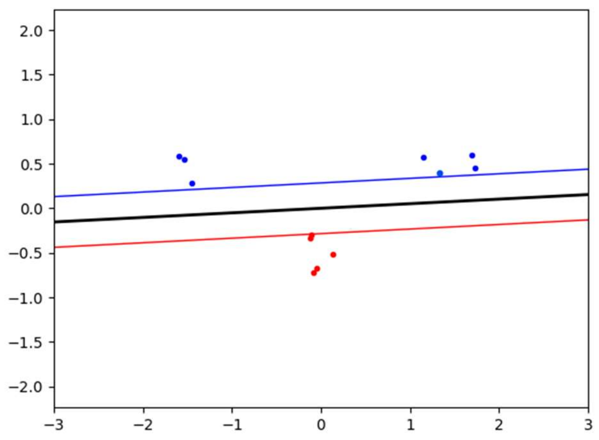
In which case/How, softs margins are interesting?

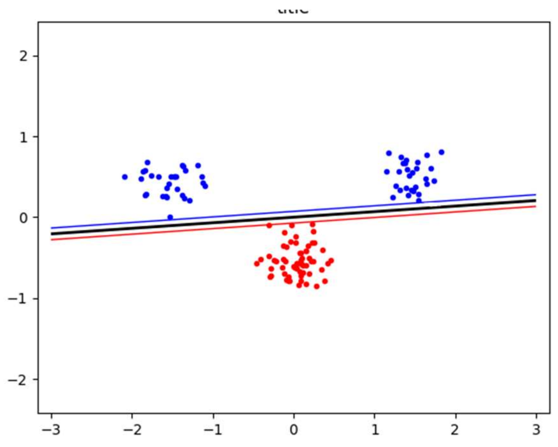
In the classification above, we have classes very distinguished, to the point where it's easy to create our own rough decision boundary. In all cases before, without slacks variables, the generating test data of the subject means that even if we were to generate new class points and we will compute the estimate class, it will be necessary in the right class, that means 0% of errors. We can say roughly that the test data is representative to a very low complexity.

On the contrary, in more complex situation, slacks variables are very useless to find an adapted classification with better results than without. We will see the impact/performance of slacks variables in the next part who represent a more complex problem.

Evolution of the slack parameter C

The evolution of the slack parameter regarding the quantity of data is important to understand which value is interesting to use. For this exploration, we need to use a different threshold value.





Linear kernel with few samples *Linear kernel with lot of samples*

Size of set	**12**	**40**	**120**
Threshold value used	0.000001	0.00001	0.1
Sum alpha values	~12.4	~24.0	~138300.0
Max alpha values	6.19	12.04	45423.03
% zero-alpha values	10/12 = 83%	36/40 = 90%	105/120 = 87%

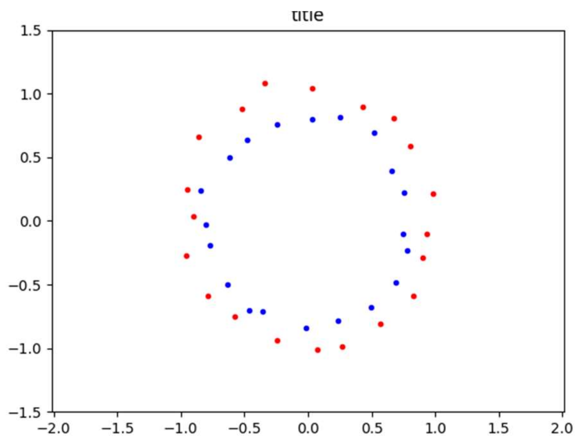
The percentage of zero-alpha values is approximately the same, but the maximum of alpha value associated to a unique SV increases exponentially.

For a same similar problem with same outliers' treatment, the problem with more samples needs a slack parameter higher because it limits the max value authorize by alpha values.

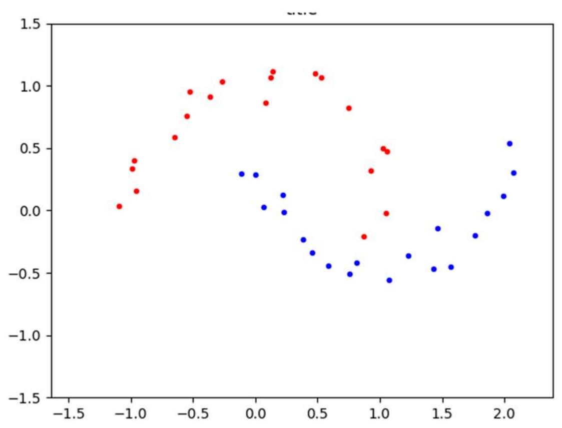
5. To find out more about classification with SVM

5.1 More complex problem

To study the impact of parameters of nonlinear-kernel functions, it's possible to use a more complex problem. To generate more complex data/samples, we use the package sklearn.dataset (Scikit learn is one of the most famous traditional tool for machine learning applications (with Tensorflow and especially)). These samples with more complex shapes are classified. In this part, we are going to study the influence of parameters in the bias-variance trade-off.



make_circle() function set

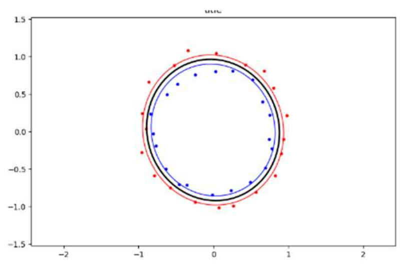


make_moon() function set

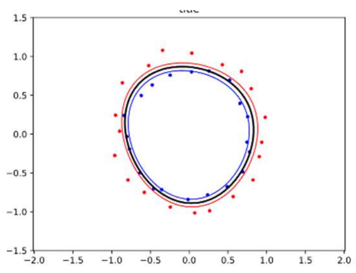
5.2 Influence of parameter in the bias-variance trade-off

Polynomial dimensionality

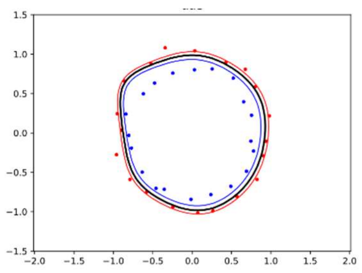
In the special case of the polynomial kernel function, we use a circle set with noise. We can already predict that dimension 2 will be a good option, as it's more of a circle shape. We use a test of 1000 samples.



p=2



p=3

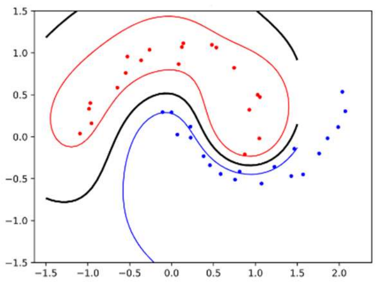


p=6

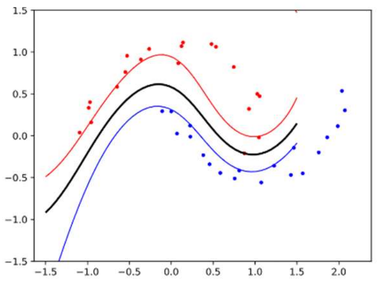
Polynomial dimension	1	2	3	4	5	6
Percent of error on testing set	No solution	0.867	0.787	0.725	0.714	0.714

RBF smoothness

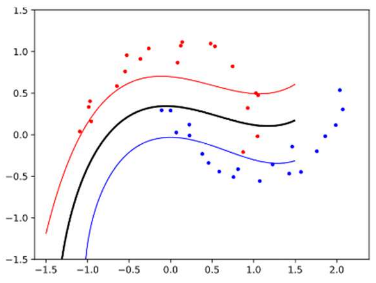
In the special case of rbf kernel function, we use a moon set with noise. As before, we use a test of 1000 samples.



Sigma=0.6



Sigma=1.0

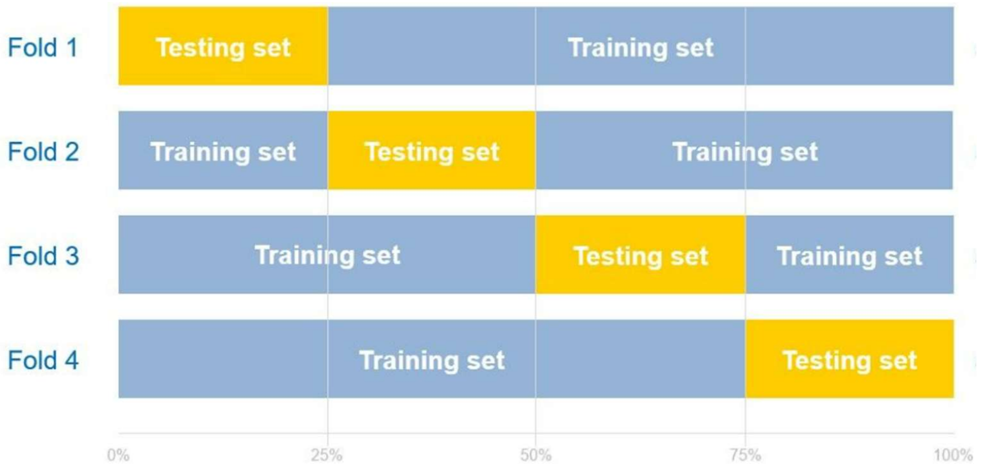


Sigma=2.0

Value of smoothness	0.6	0.8	1.0	1.3	2.0	3.0
Percent of error on testing set	0.979	0.990	0.994	0.979	0.942	0.843

Possibly usage of cross-validation

One way of studying a model using hyperparameters allowing one training set and a testing set can be to use cross-validation. This method was see during lectures and can be a good method to implement in this laboratory

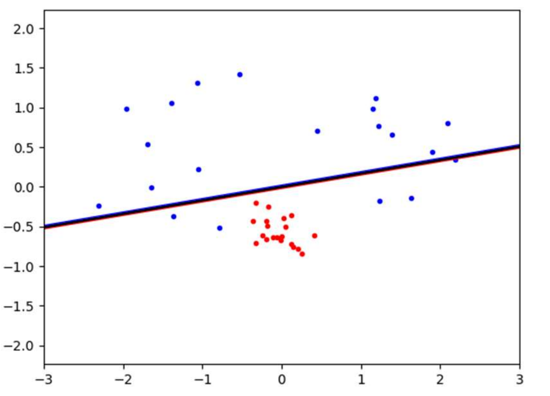


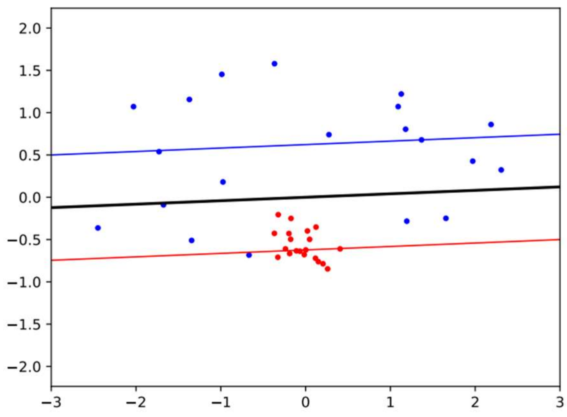
Usage of cross-validation schema

5.3 Limits of the optimizer

Unabled to find solution and balance slack/complexity

One problem of the optimizer proposed by *spciy (function minimize)* is the very bad computation if it's not possible to find a solution at all.



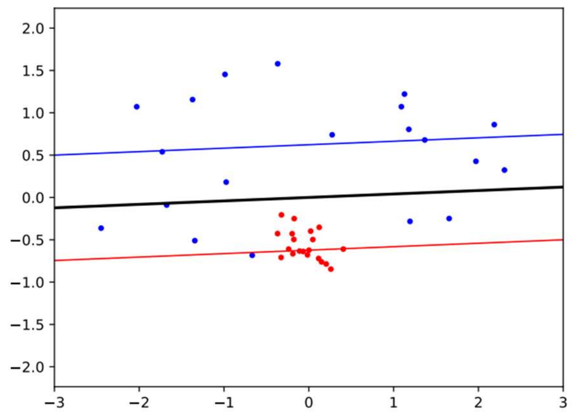


Linear kernel without slack-variables

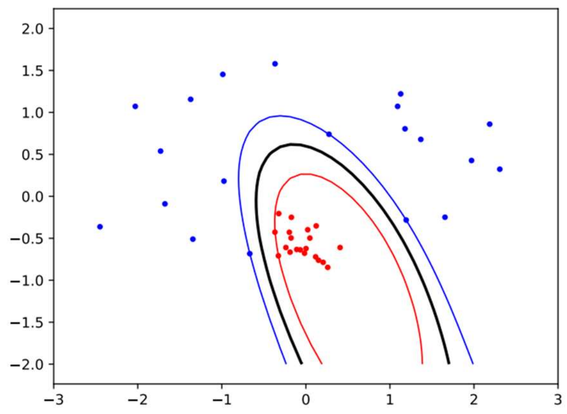
Linear kernel with slack-variables (C=0.5)

To understand the limit of the optimiser, we use a linear kernel function (obviously that's not the best kernel function to choose at all in this case) on a widely distributed dataset. As we can see on the left plot (without slack-variables), the minimize function does not find a solution (error message: Iteration limit reached). Alpha values are not restricted, and no SV are found. On the contrary, the right plot looks better, even if there are some training samples misclassified because a solution is possible to find. That's because slack-variables are used.

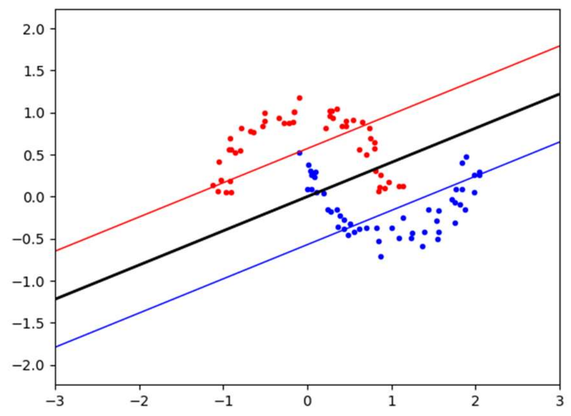
Used a model without understanding its concepts or its parameters can lead to very bad classifiers models.



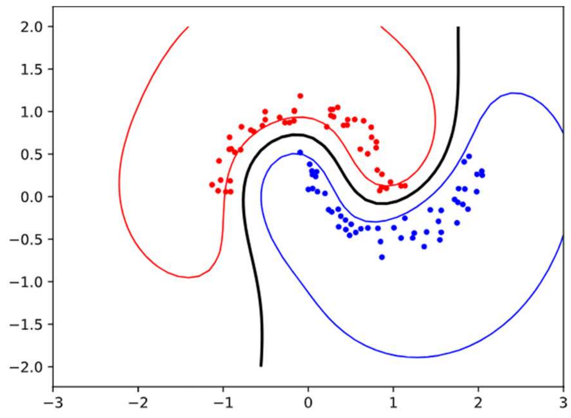
Linear kernel with slack-variable (C=0.5) less adapted



RBF kernel without slack-variable (sigma=2) more adapted



Linear kernel with slack-variable (C=0.5) less adapted



RBF kernel without almost slack-variable (sigma=0.8) more adapted

In the case of plots above, again it's obviously (graphically and easy to prove it with calculation with different methods using test set) that we should opt for a more complex model kernel than linear kernel (such as polynomial or RBF kernel) than opt for more slack because the shape of the linear kernel meaning line are not adapted.

6. Conclusion

To conclude about this exploration laboratory on the specific subject of classification using Support Vector Machines model, we can respond to the following reflections:

1. Reaction of the model/optimizer when it's not able to find a solution

Saying that the model is not able to find a solution it's equivalent to say that the optimizer is not able to emphasize any SV. So, no boundaries are possible to find to separate classes. A typical example of a model not able to find a solution is two classes with outliers respectively each in the other with a linear kernel function and no slack variable.

2. Implementation of non-linear kernels and impact on classification

Non-linear kernels are very practical when the data represents a complex problem. Sometimes, the classification needs boundaries not linear but curved etc. The classification can be better when a correct kernel function is used. In this laboratory we implement polynomial kernel function and rbf kernel function. In both these cases, hyperparameters allow us to obtain different model of classification more or less correct: more possibilities are offered in exchange for a more complex study (usage of cross validation, reflection on bias-variance trade-off and etc.)

3. Influence of parameters of non-linear kernels in terms of the bias-variance trade-off

Depending on the nonlinear kernel functions, parameters influence the bias-variance trade-off. They can be named hyperparameters.
- In the case of polynomial kernel function: increase the dimensionality (*p*) increase the variance (implies reduce of bias). It means that the overfitting effect is more present with a high dimensionality. Indeed, increase the dimensionality is equivalent to increase the complexity of our model.
- In the case of rbf kernel function: increase the smoothness of the boundaries (sigma) will lead of a lower complex model and so increase the bias of the model (implies decrease the variance). It means that the overfitting effect is more present with a lower value. On the contrary a higher value will lead to the underfitting effect because of a too general model unable to capture all information/shapes.

4. Exploring slack parameter C in large/small values (increase/decrease value)

For a similar problem of classification, bigger the set used to train the model, bigger the slack parameter C must maintain consistency.

Lower the slack parameter C is, bigger the SV are constraint, and more errors are authorized. We can consider that reducing the slack parameter reduce the overfitting effect (implies reduce of variance). Lower the slack parameter, more SV are involved (means more non-zeroes alpha values).

It is important to remember that variable slacks are a good way of dealing with outliers.

5. Balancing between opt for more slack or for more complex model

After exploring different ways, it's not obvious to opt for more slack or for more complex model, it really depends on different things: the shapes of classes use for training the model and the complexity wanted by our model (linear does not use hyperparameter). If the separation of classes does not seem linear, it's preferable to use another kernel function. As a reminder, in some cases, the optimizer was not able to find solution because the classes were absolute not linearly usable. In many cases, opt for a more complex model is better because it is more common to study a phenomenon not linear. However, this notice is only based on my few different experiences in machine learning.

Globally, we can say that SVM model is good at finding a reasonable decision boundary from small sets of training data thanks to variety of kernel functions that can be used and the possible regularization using slack variables. Moreover, small sets are preferable because optimization takes time to obtain even with optimize computation (numpy sum, precomputable matrix, etc.).