# ECE 8443: Pattern Recognition – Project 2

Support Vector Machines

Ruperto Solis – Mississippi State University

*ABSTRACT*

***This document contains the steps for the development of a two-class support vector machine classifier for linearly and non-linearly separable data, including the design from the geometrical interpretation of the perceptron and the optimization of the tool through the maximization of the margin from the decision hyperplane and the minimization of the misclassification of the data. Here you can find the methodology for the implementation of different kernels such as the radial base function, inner product and quadratic polynomial with their respective combinations to build multi-kernel approaches. The designed SVM classifier is subjected to four different experiments with different data distributions, depicting the superiority of some kernel combinations with respect to others given the specific applications, and the contrast of the SVM with respect to the minimum risk Bayes theoretic classifier in terms of flexibility, accuracy and processing time from a qualitative perspective.***

## I. INTRODUCTION

In machine learning and pattern recognition, the support vector machines are very flexible supervised learning models for classification, capable of providing different solutions for a single classification problem, by allowing a user to modify several input parameters. SVM can provide approximations for non-linearly separable applications by solving the problems in a linear fashion in spaces of higher dimensionality. Since this tool is a non-probabilistic binary linear classifier, previous knowledge of probabilistic density functions is not required, relying instead on the power provided by its capability of implementing different kernel combinations depending on the application.

## II. THEORY BEHIND THE TWO CLASS SUPPORT VECTOR MACHINE CLASSIFIER

### The perceptron

To understand the functionality of the support vector machine it is necessary to start with the definition of the perceptron algorithm, since in its most essential and primitive form, the support vector machine is a mathematical model created to compute the components that characterize the most favorable decision hyper-plane for the perceptron.

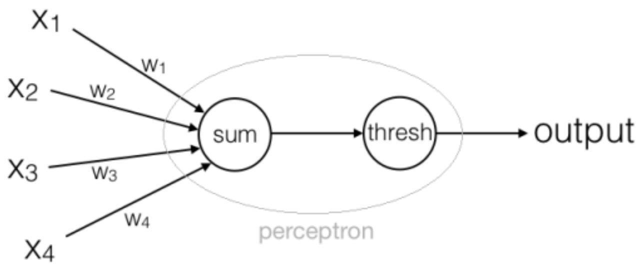


Figure 1 – Perceptron structure

In deep learning, the perceptron is the mathematical representation of the biological neuron whose functionality is in function of a series of input values, their weighted summation, and a threshold value that stablishes the minimum value the weighted summation needs to achieve in order to trigger an output $\sum w_i x_i \geq -w_o$. In pattern recognition, the perceptron has a geometric interpretation: We are going to visualize a classification case in a 2-dimensional Cartesian plane, because is more intuitive that way and can be easily expanded to higher dimensions, where there are 2 classes and they are nicely separated by a line known as the decision line or decision hyper-plane in the multidimensional interpretation. This line is the geometric representation of a discriminant function created using the perceptron inequality:

$$g(\vec{x}) = \vec{w}^T \vec{x} + w_o. \qquad (1)$$

Where $\vec{w}$ is the normal of the decision line, $\vec{x}$ is the position vector of any point in the Cartesian plane to be classified, and $w_o$, in combination with the vector $\vec{w}$, is used to determine the bias for each of the axis as described in Figure 2.

If we subject every point of the Cartesian plane to the discriminant function defined in equation 1, we find that every point on the decision line has a value $g(\vec{x}) = 0$, $g(\vec{x}) > 0$ for every point on the Class 1 side of the line and $g(\vec{x}) < 0$ on the non-Class 1 side. This means that when we have appropriate values for vector $\vec{w}$ and $w_o$ we can use this discriminant function to successfully classify any point in the plane as class 1 when $g(\vec{x}) \geq 0$ and non-Class 1 otherwise. As it was mentioned before, we can expand this classification into higher dimensional spaces by using the same discriminant function; the only difference with respect to the 2 dimensional case, is that vector $\vec{w}$ will have one extra component for every additional dimension. For the D dimensional space $\vec{w} = [w_1 \;\; w_2 \;\; w_3 \; \dots \; w_D\,]$ and as a result, the decision hyper-plane will be a D-1 dimensional object (e.g. a plane in the 3 dimensional space).

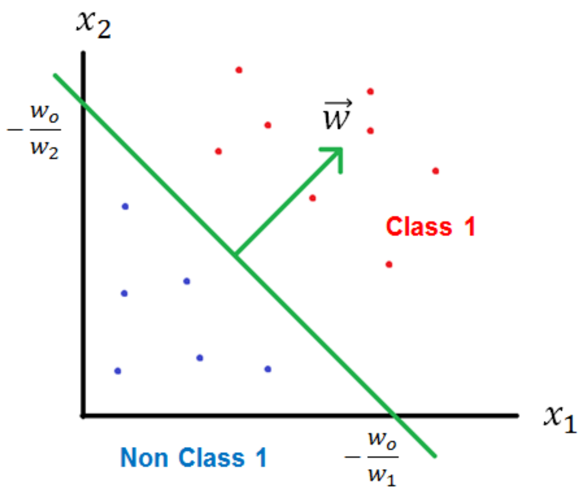


Figure 2 – Decision hyper-plane.

**Calculating $\vec{w}$ and $w_o$**
Here is where the support vector machine comes into play, it is a supervised learning technique that allow us to design the decision hyper-plane on a linearly separable classification problem as an optimization model where the two closest points from opposite classes are used as supports and the decision hyper-plane is placed right in between the support vectors, orthogonal to the distance vector of these points. We can identify as $X$, the entire collection of all the samples in the space, including all class 1 and non-class 1 samples. We can also use $y$ as a column vector with the labels of the entire collection $X$, where all the elements that belong to class 1 have a value of $y(i) = 1$, and $y(j) = -1$ for any non-class 1 element.

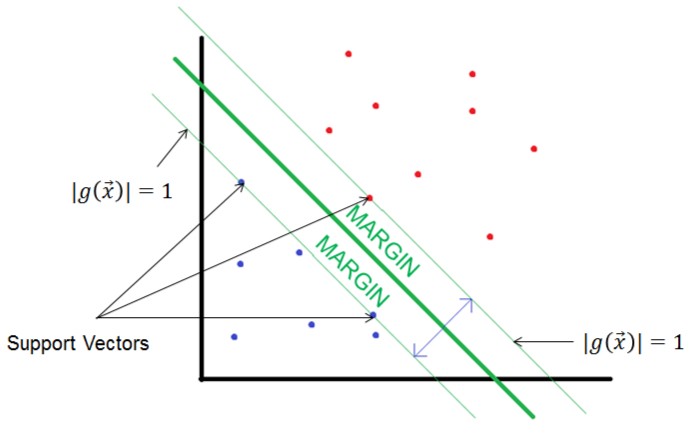


Figure 3 – Margin and support vectors.

The closest Euclidean distance of any given point in the space with respect to the decision hyper-plane $Z$ is defined in equation 2 as a normalized version of the discriminant function with respect to vector $\vec{w}$.

$$Z(g(\vec{x})) = \frac{g(\vec{x})}{||\vec{w}||} \qquad (2)$$

Hence, the closest distance between the limit hyper-planes of each class, located at $|g(\vec{x})| = 1$, known as the margin is:

$$margin = \frac{2}{||\vec{w}||} \qquad (3)$$

In a linearly separable classification problem, there are probably an infinite number of solutions, each defining its own decision hyper-plane. However, using the information from the support vectors, we will fall under the assumption that the best solution corresponds to the hyper-plane that can provide the largest margin while maintaining a $g(\vec{x}) = 1$ (for convenience) at the support vectors. That way this becomes an optimization problem where we want to maximize the margin. An equivalent version of the maximization problem described before is shown in equation 3, converting it into a minimization problem with a discriminant cost function of $||\vec{w}||$ (making it easier to solve).

$$J = \frac{1}{2}\ ||\vec{w}||^{\ 2} \quad (4)$$

Subjected to $y(\vec{w}^T X + w_o) \geq 1$

Where $y$ is the labels column vector and $X$ is the entire samples collection. This means that in the minimization problem we have 1 constraint for every sample for which $g(\vec{x})$ needs to be larger or equal to one. For the non-class 1 samples, the fact that $y(i) = -1$ helps to fool the constraints equation making it think that these elements are on the class 1 side of the hyper-plane, making possible to really have all the constraints as >=1.
If we solve using the Lagrange multipliers optimization technique, using its primal form we can discover that there are new conditions, known as KKT conditions (equations 6 and 7 below), which we need to meet in order to comply with the requirements of the cost function and the initial constraints.

Lagrange function:

$$L(\vec{w}:, w_o, \lambda) = \frac{1}{2}||\vec{w}||^{\ 2} - \lambda[y(\vec{w}^T X + w_o) - 1] \qquad (5)$$

KKT conditions, obtained after doing partial derivatives w.r.t. $w, w_o$:

$$\vec{w} := \textstyle\sum_{i=1}^{N} \lambda_i y_i \vec{x}_i \qquad (6)$$

$$\textstyle\sum_{i=1}^{N} \lambda_i y_i = 0 \qquad (7)$$

When combining equations 4, 6 and 7 we obtain the dual form of the Lagrange which is a maximization function of $\lambda$:

$$maximize \quad \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \vec{x}_i^{\,T} \vec{x}_j \quad (8)$$
$$\text{Subjected to} \quad \sum_{i=1}^{N} \lambda_i y_i = 0 \quad \text{and} \quad \lambda \geq 0$$

In its dual form, math wise, it is a very simple problem to solve, the only variables in the function are now the Lagrange multipliers. Both $y$ and $X$ are constant for each specific classification problem. The only inconvenient is that it can be a very large and exhausting equation system to solve; the cost function involves summations of order O(N) on the first term and O(N^2) on the second one. In addition to that, the cost function needs to be subjected to N partial derivatives, one for each Lagrange multiplier, creating a multivariable equation system of N variables and N equations. Since every classification problem has different values for $y$ and $X$, there's no unique solution. This leads us to rely on the power of computation; many programming languages have embedded functions to solve this type of quadratic problems and there are several open source libraries in the web dedicated to implement this functionality in popular programming languages.
The output of the quadratic programming functions is a vector containing the values of all the Lagrange multipliers $\lambda$. Every multiplier $\lambda_i$ is associated to a sample $\vec{x}_i$, and the value of $\lambda_i$ will determine whether or not the vector $\vec{x}_i$ is a support vector; $\vec{x}_i$ is considered a support vector if $\lambda_i > 0$, otherwise $\lambda_i = 0$.

***Now we have enough information to calculate what we were looking for: $\vec{w}$ and $w_o$ :***
We can use equation 6 to directly calculate the $\vec{w}$:

$$\vec{w} := \sum_{i=1}^{N} \lambda_i y_i \vec{x}_i$$

Using equation one and the fact that the support vectors are on the edge of the margin, meaning that $|g(\vec{x})| = 1$, we can solve for $w_o$:

$$w_{o_i} = \frac{1}{y_i} - \vec{w}^T \vec{x_i} \quad (9)$$

Since there may be several support vectors, it is a good idea to calculate $w_o$ as the average of all $w_{o_i}$.

**Classification**
Now that we obtained the components of the decision hyper-plane $\vec{w}$ and $w_o$ , we can plug them back in the discriminant function described in equation 1:

$$g(\vec{x}) = \vec{w}^T \vec{x} + w_o$$

Here are all the possible outcomes for $g(\vec{x})$ depending of which vector $\vec{x}$ is evaluated:

| $g(\vec{x})$ | Classification | Notes |
|---|---|---|
| $g > 1$ | **Class 1** | |
| $\mathbf{1 \geq g > 0}$ | **Class 1** | Vector is in the margin |
| $\mathbf{g = 0}$ | **Class 1** | Vector is right on the decision line |
| $\mathbf{-1 \leq g < 0}$ | **Non Class 1** | Vector is in the margin |
| $\mathbf{g < -1}$ | **Non Class 1** | |

Table 1 – Classification results under discriminant function $g(\vec{x})$.

In its most basic form, the SVM is a 2 class multidimensional classifier; however it can be easily expanded to multiclass by a series of different techniques such as creating an individual SVM for each class or a multi-layer combination, which are beyond the scope of this report.

**Linear solution for non-linear cases**

In some cases, the data to classify can have noise or even certain level of similarity between classes at the margin that will make it impossible to separate with a simple linear hyper-plane in the D dimensional Euclidean space, as exemplified on figure 4.

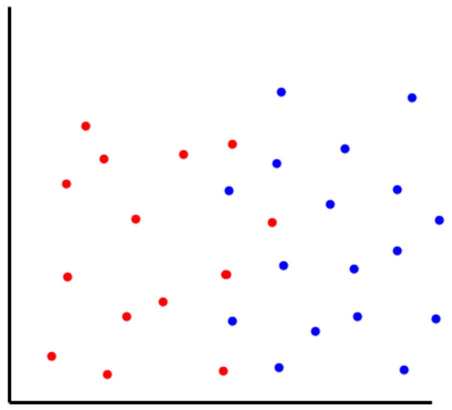


Figure 3 – Non linearly separable data.

As we can see on figure 3, it is not possible to separate the blue vectors from the red vectors using a single line, however, we can do the best we can and find a line that can "almost" separate the two classes producing only a few errors during the classification. In order to do this we need to introduce a new term to our cost function with the intention of minimizing the classification error. We can tell if there is an error if when subjecting a vector to the "regulated" discriminant function, it returns a value less than zero: Error if $y_i g(\vec{x}_i) < 0$. To represent the error, we can introduce the new variable $\xi_i$ which is an indicator of the error:

$$y_i g(\vec{x}_i) \geq 1 - \xi_i$$

Hence, if there is an error $\xi_i > 1$

We can then produce the following cost function accounting for all the error indicators as a minimization problem:

$$J = \frac{1}{2}\, \|\vec{w}\|^{2} + c \sum_{i=1}^{N} \xi_i \qquad (10)$$

Subjected to $\quad y_i(\vec{w}^T x_i + w_o) \geq 1 - \xi_i \quad \xi_i \geq 0$

Following the same steps as in the linearly separable problem (solve the primal, get KKT conditions, formulate the dual form of the Lagrange), turns out that we have the same optimization problem, except for an additional constraint: $c \geq \lambda_i$.

$$maximize \quad \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \vec{x}_i^{\,T} \vec{x}_j \quad (11)$$

Subjected to $\quad \sum_{i=1}^{N} \lambda_i y_i = 0 \quad$ and $\quad c \geq \lambda \geq 0$

At this point nothing changes, the classification is done through computing of a quadratic program just as in the linearly separable case, the only difference will be the inclusion of factor c, which help us to control the “importance” of the minimum error term in the cost function; the highest the value of c, the more importance we are giving to minimize the error w.r.t. maximizing the margin.

**Non-linearly separable classifier using Kernels**

By maximizing $\lambda$ in equation 11, we can provide the best linear solution which is when the classification error is minimized. However, in some cases the classes have very particular distributions or our allowance for error is so tight that the errors obtained by using this method are just way too many to be acceptable. Here is where we introduce the concept of kernel.

The kernel is an N by N function matrix where each element is a function of 2 vectors in the collection of samples X. There is 1 element in the matrix for each possible combination of $x_i$ and $x_j$. For example, the most basic type of Kernel used in Pattern Recognition is the dot product kernel: $K_d = X^T X$.

Turns out that our maximization function is already using the dot product kernel in the Lagrange function, it is what gives it the linear behavior in the D dimension Euclidean space. Different kernels provide different behaviors and can take our D dimensional data to much higher Dimensional spaces where non-linearly separable data becomes linearly separable. Which kernel to use will depend on the type of distribution you have on the data, you can even combine the kernels by performing a weighted summation of different kernels and experiment with a variety of solutions to identify which is the best for your specific application.

| Kernel | Function |
|---|---|
| Dot Product | $K_d(x_i, x_j) = x_i \cdot x_j$ |
| Quadratic | $K_q(x_i, x_j) = (x_i \cdot x_j + 1)^2$ |
| RBF (Radial Base Function) | $K_r(x_i, x_j) = \exp\left(-\frac{\lVert x_i - x_j \rVert^2}{2\sigma^2}\right)$ |

Table 2 – Some Kernels in SVM.

In order to use a different kernel, the only things that need to be done are:

1) Replace the dot product kernel for any kernel or combination of kernels prior to the quadratic programming computation:
2)

$$\sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j K(x_i, x_j) \qquad (12)$$

3) As a result of this substitution, we need to replace $\vec{x}_i^{\,T}\vec{x}_j$ by $K(x_i, x_j)$ everywhere, including the equations used to calculate $w_o$ and finally our discriminant function $g(\vec{x})$.
   By combining equations 6 and 9 we get:

$$w_{o_i} = \frac{1}{y_i} - \sum_{j=1}^{N} \lambda_j y_j K(x_i, x_j) \qquad (13)$$

Again $w_o$ is calculated as the average of all $w_{o_i}$.

By combining equations 1 and 6:

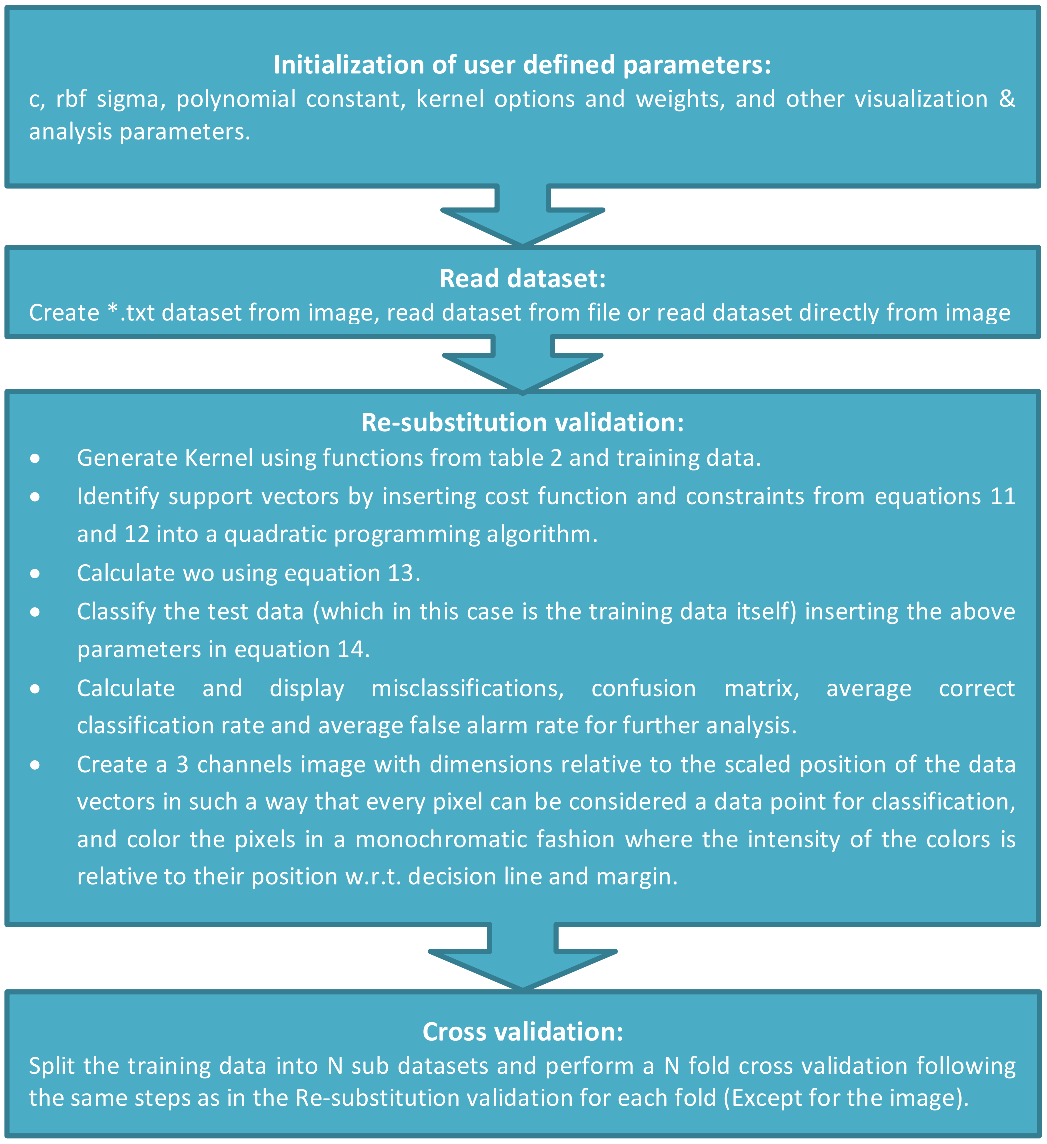
$$g(\vec{x_i}) = \sum_{j=1}^{N} \lambda_j y_j K(x_i, x_j) + w_o \qquad (14)$$

## III. DESIGN OF A 2 CLASS SVM CLASSIFIER IN MATLAB

It's time to implement the theory from the previous section to design a 2 class SVM classifier. The following classifier will provide to the user, the ability to choose from 4 different kernels (dot Product, quadratic polynomial, cubic polynomial and radial base function) and any combination of them with their respective weights and some internal parameters such as sigma in the rbf kernel and the constant in the polynomial kernels. In addition, the user will be able to play with 'c' from equation 10 to control the 'importance' of minimizing misclassification with respect to the maximization of the margin.

The structure of the program consists on a main 'm' file called Main_Project2.m which will be calling other functions distributed in different files for ease of reading and understanding of the methodology.

The following pipeline describes the functionality of the Main_Project2.m:



**Initialization of user defined parameters:**
c, rbf sigma, polynomial constant, kernel options and weights, and other visualization & analysis parameters.

**Read dataset:**
Create *.txt dataset from image, read dataset from file or read dataset directly from image

**Re-substitution validation:**

- Generate Kernel using functions from table 2 and training data.
- Identify support vectors by inserting cost function and constraints from equations 11 and 12 into a quadratic programming algorithm.
- Calculate wo using equation 13.
- Classify the test data (which in this case is the training data itself) inserting the above parameters in equation 14.
- Calculate and display misclassifications, confusion matrix, average correct classification rate and average false alarm rate for further analysis.
- Create a 3 channels image with dimensions relative to the scaled position of the data vectors in such a way that every pixel can be considered a data point for classification, and color the pixels in a monochromatic fashion where the intensity of the colors is relative to their position w.r.t. decision line and margin.

**Cross validation:**
Split the training data into N sub datasets and perform a N fold cross validation following the same steps as in the Re-substitution validation for each fold (Except for the image).

## Key Algorithms

### *Algorithm 1 – Read image file and export as dataset*

```
function [DataSet,labels] = RBImageToDataset()
    [FileName,Path]=uigetfile({'*.*'});
    imageDir=[Path,FileName];
    ImageM=imread(imageDir);
        R=ImageM(:,:,1);
        G=ImageM(:,:,2);
        B=ImageM(:,:,3);
            Red=R-B;
            Red=Red>25;
            [Redy,Redx]=find(Red);
            Blue=B-R;
            Blue=Blue>25;
            [Bluey,Bluex]=find(Blue);
    BlueData=[Bluex,Bluey];
    RedData=[Redx,Redy];
    DataSet=[RedData;BlueData];
    labels=[ones(size(RedData,1),1);-1*ones(size(BlueData,1),1)];
```

### *Algorithm 2 - Generate kernel*

```
function kernel = getKernel(DataSet,DataSet2,kernelOptions,kernelWeights,rbf_sigma,poly_Const)
          N=size(DataSet,1);
          N2=size(DataSet2,1);
        %Get H based on Kernels selection:
            lenk=size(kernelOptions,2);
            kernelOptions=kernelOptions(2:1:lenk);
            lenkw=size(kernelWeights,2);
            kernelWeights=kernelWeights(2:1:lenkw);
            KArray=strsplit(kernelOptions,'-');
            KWArray=str2double(strsplit(kernelWeights,'-'));
            %Normalize weights
            KWArray=KWArray./sum(KWArray);
            K=zeros(N,N2);
                %Dot Product Kernel (Inner Product)
                    Kindex=find(strcmp(KArray,'dot'));
                    if size(Kindex,2)>0
                        K=K+KWArray(Kindex)*(DataSet*DataSet2');
                    end
                %RBF Kernel (Radial Based Function)
                    Kindex=find(strcmp(KArray,'rbf'));
                    if size(Kindex,2)>0
                        dimensions=size(DataSet,2);
                        tempMatrix=zeros(size(DataSet,1),size(DataSet2,1),dimensions);
                        for i=1:dimensions
                          tempMatrix(:,:,i)=DataSet(:,i)*ones(1,size(DataSet2,1));
                        end
                        tempMatrix2=zeros(size(tempMatrix));
                        for i=1:dimensions
                          tempMatrix2(:,:,i)=ones(size(DataSet,1),1)*DataSet2(:,i)';
                        end
                        subsMatrix=(tempMatrix2-tempMatrix).^2;
                        addMatrix=zeros(size(DataSet,1),size(DataSet2,1));
                        for i=1:dimensions
                          addMatrix=addMatrix+subsMatrix(:,:,i);
                        end
                        addMatrix=exp(-0.5.*addMatrix./(rbf_sigma)^2);
                        K=K+(KWArray(Kindex).*addMatrix);
                    end
                %POLY2 (QUADRATIC POLYNOMIAL)
                    Kindex=find(strcmp(KArray,'poly2'));
                    if size(Kindex,2)>0
                        K=K+KWArray(Kindex)*(((DataSet*DataSet2')+poly_Const).^2);
                    end
                %POLY3 (QUADRATIC POLYNOMIAL)
                    Kindex=find(strcmp(KArray,'poly3'));
                    if size(Kindex,2)>0
                        same way to obtain submatrix 1 and 2 as in the rbf.
                        subsMatrix=(tempMatrix2+tempMatrix)+poly_Const;
                        same way to generate the addMatrix as in the rbf before the exponential.
                        addMatrix=addMatrix.^3;
                        K=K+(KWArray(Kindex).*addMatrix);
                    end
                kernel=K;
```

## *Algorithm 3 – Identify support vectors and calculate Wo*

```
function [Wo_avg,Indexes,alpha] = nonlinear_SVM(labels, kernel, c,  minAlpha)
        %Paramters for the quadratic programming
        N=size(labels,1);
        H=kernel.*(labels*labels');
        %Rest of parameters of Quadratic Programming
        f=-ones(N,1);
        %Inequality constraints
          A=eye(N);
          lb=zeros(N,1);
          ub=c*ones(N,1);
          b=ub;
        %Equality Constraints
        Aeq=[labels';zeros(N-1,N)];
        beq=zeros(N,1);
        %Quadratic Programming
        alpha=quadprog(H+eye(N)*0.001,f,A,b,Aeq,beq,lb,ub);
        Indexes=find(alpha>minAlpha);
        Wo_avg=0;
        totSV=size(Indexes,1);
        for i=1:totSV
          index=Indexes(i);
          Wo=1/labels(index,1)-sum(alpha.*labels.*kernel(:,index));
          Wo_avg=Wo_avg+Wo;
        end
        Wo_avg=Wo_avg/totSV;
```

## *Algorithm 4 – Generate Background Image with 'classified' pixels*

```
function img =
backImage(Xscale,Yscale,imgDataSet,supportVectors,SValpha,SVlabels,Wo,KernelOptions,KernelWeights,rbf_s
igma,poly_Const) %imgDataSet is a scaled version of the dataset to match image dimensions
            minx=floor(min(imgDataSet(:,1)));maxx=ceil(max(imgDataSet(:,1)));
            miny=floor(min(imgDataSet(:,2)));maxy=ceil(max(imgDataSet(:,2)));
            xAxis=minx:1:maxx; yAxis=(miny:1:maxy)';
            rows=maxy-miny+1; cols=maxx-minx+1;
            xyMatrix(:,:,1)=ones(rows,1)*xAxis; xyMatrix(:,:,2)=yAxis*ones(1,cols);
            pixels=[];
            for i=1:rows
              rowElements=zeros(2,cols);
              rowElements(1,:)=xyMatrix(i,:,1); rowElements(2,:)=xyMatrix(i,:,2);
              pixels=[pixels;rowElements'];
            end
            eqPixels=[pixels(:,1)./Xscale,pixels(:,2)./Yscale];
            kernel=getKernel(supportVectors,eqPixels,KernelOptions,KernelWeights,rbf_sigma,poly_Const);
            g=((SValpha.*SVlabels)'*kernel)+Wo;
            gmatrix=zeros(rows,cols);
            for i=1:rows
              rowElements=g(1,counter:1:counter+cols-1);
              gmatrix(i,:)=rowElements;
              counter=counter+cols;
            end
            %Dark red on red margin
                redBack1=60.*(gmatrix>0);
                blueBack1=60.*(gmatrix<0);
            %Further than the Margin
                redBack2=180*((gmatrix>=1).*gmatrix)/max(max(gmatrix));
                blueBack2=180*((gmatrix<=-1).*gmatrix)/min(min(gmatrix));
            %Add Reds/Blues
                redBack=redBack1+redBack2;
                blueBack=blueBack1+blueBack2;
            %Green Decision Line
                greenBack1=gmatrix>=-0.05; greenBack2=gmatrix<=0.05;
                greenBack=and(greenBack1,greenBack2)*255;
            %Not as bright Margin line
                greenBack1=abs(gmatrix)>=0.9; greenBack2=abs(gmatrix)<=1.1;
                greenBack=greenBack+and(greenBack1,greenBack2)*125;
            %Cap at 255
            redBack=((redBack>255).*255)+((redBack<=255).*redBack);
            blueBack=((blueBack>255).*255)+((blueBack<=255).*blueBack);
            greenBack=((greenBack>255).*255)+((greenBack<=255).*greenBack);
            BackColor(:,:,1)=uint8(redBack);
            BackColor(:,:,2)=uint8(greenBack);
            BackColor(:,:,3)=uint8(blueBack);
            img=BackColor;
```

## IV. RESULTS AND CONCLUSIONS

During the development of the algorithms I had the opportunity to discover little by little the power of the Support Vector Machine. Initially, it was hard for me to understand how the non-linear cases were going to be addressed, since I wanted to stick to the definition of vector W and bias Wo as the components of a hyperplane in the D-dimensional Euclidean space. Another component in this collective confusion occurred during the introduction of the kernels, making me think that changing a kernel was as easy as generating it and just plugging it into the dual of the Lagrange equation to obtain the support vectors. Then I was going to be able to calculate W and Wo in the same way as I would do for the linearly separable cases. It wasn't until I solved the equations myself that I realized that W was in function of the inner product as well and if I was going to change the dot product kernel, then I would need to come up with a different way to introduce W into the discriminant function g(x), as I did in equation 14. Something I had to discover in the hard way was that when performing the quadratic programming, things are not as dogmatic as they are in paper. In most cases the results of our optimization problem didn't perfectly follow the constraints providing values of lambda slightly higher than 0 for the non-support vectors, that's where I introduced the minAlpha parameter which I used as a tress hold for identifying the true support vectors.

The SVM classifier was tested on a series of 4 experiments, please refer to Appendix A for the complete list of results from all 4 experiments. Following the requirements of the project, I created a few nonlinearly separable 2D datasets, including an image with class imbalance, and found a dataset with 100 features in the UCI website called 'Hill-Valley Data Set'. During the experiments, I used a combination of different kernels to test different solutions and compared my results those of the minimum risk Bayes theoretic classifier which led me to the following conclusions:

- The Gaussian radial base function is a very powerful kernel, once you get to play with it for a while you learn that it starts by identifying each vector as its own class with certain radial distance, and by adjusting sigma, such distance, which for the moment I am going to call 'field', is regulated to a point where when fields of two vectors of the same class encounter, they merge creating larger fields. This is very powerful because if there is a well-defined proximity between vectors of the same class and they are decently separated from vectors from other classes, you can basically create generic classifiers for almost any type of distribution including those with complex shapes.
- The Gaussian kernel is very noise sensitive. Noise in the data can create very strange shapes that will mess up with the generic classifications. in some cases, although not 100% accurate, the inner product kernel and the quadratic kernel can create more generic classifiers as I found out during the N folds' cross validation, where they proved that not memorizing every single vector can lead to superior classifiers in the long run by using less support vectors.
- With respect to its performance against Bayes, when it comes to very large data, the SVM algorithm can be very slow w.r.t. the Bayes classifier from a qualitative perspective, the number of complex operations and the size of the resultant kernels can really make the difference. Although Bayes can be an optimal classifier, you need to know a lot about the data before classifying, since the use of the most suitable pdf is the key for its success. With the right visualization tools, SVM can be much more flexible tool.

## APPENDIX A

On the red/blue visualization images, every pixel was colored based on its classification status using the following criteria:

- Red pixels belong to class 1.
- Blue pixels are non-class 1.
- Lighter red and blue tones represent relative distance from the margin.
- Dark uniform tones of red and blue represent pixels that are inside of the margin.
- Bright green pixels represent the decision line with some tolerance added for visualization.
- Lighter green pixels represent the edges of the margin with some tolerance for visualization.
- Support vectors are surrounded by a yellow square.
- Incorrectly classified vectors are surrounded by a white circle.

### EXPERIMENT 1

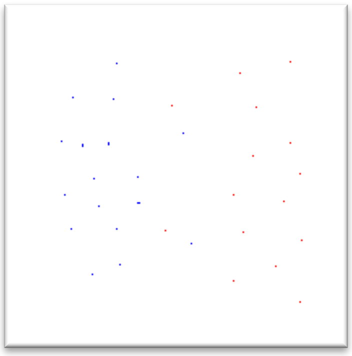


Figure A1 – Experiment 1 subject Image

Total Samples: 35 (15 red / 20 blue)

***SVM classification with dot product Kernel***

*Re-substitution classification:*

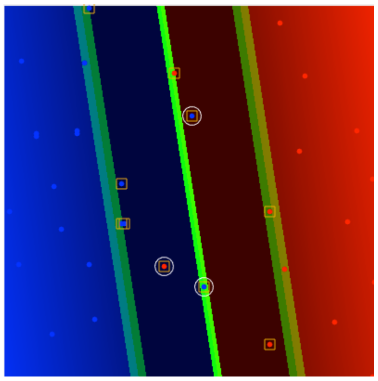


Figure A2 – Experiment 1 SVM dot product classification

<u>Classification results:</u>
CCR = 91.67%
FAR = 8.88%

Confusion Matrix:

| | Is Red | Is Blue |
|---|---|---|
| Classified as red | 14 | 2 |
| Classified as Blue | 1 | 18 |

*5 Folds cross Validation:*

| % | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
|---|---|---|---|---|---|
| CCR | 100 | 100 | 100 | 100 | 0 |
| FAR | 0 | 0 | 0 | 0 | 100 |

***SVM classification with a dot product/quadratic polynomial multi kernel***

*Re-substitution classification:*

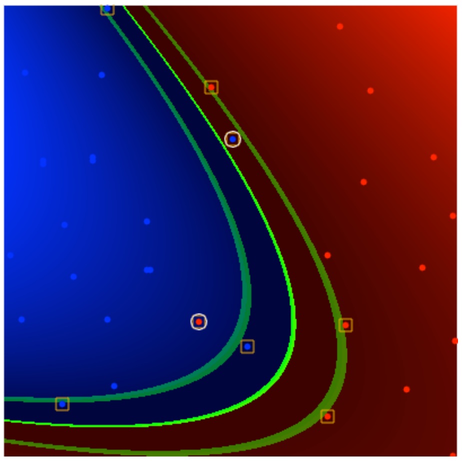


Figure A3 – Experiment 1 SVM dot product/quadratic polynomial multi kernel

<u>Parameters used:</u>

c=100

polynomial constant=10

Multi kernel weights: dot product 50%, quadratic polynomial 50%

<u>Classification results:</u>

CCR = 94.17%

FAR = 5.83%

Confusion Matrix:

| | Is Red | Is Blue |
|---|---|---|
| Classified as red | 14 | 1 |
| Classified as Blue | 1 | 19 |

*5 Folds cross Validation:*

| % | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
|---|---|---|---|---|---|
| CCR | 100 | 100 | 100 | 100 | 100 |
| FAR | 0 | 0 | 0 | 0 | 0 |

***SVM classification with a Gaussian radial based function kernel***

*Re-substitution classification:*

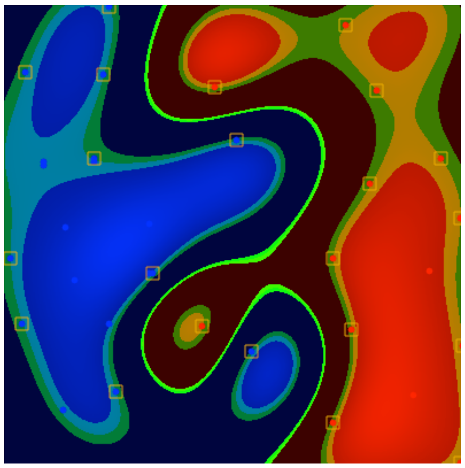


Figure A4 – Experiment 1 SVM Gaussian rbf kernel

Parameters used:

c=100

sigma=20

Classification results:

CCR = 100%

FAR = 0%

Confusion Matrix:

| | Is Red | Is Blue |
|---|---|---|
| Classified as red | 15 | 0 |
| Classified as Blue | 0 | 20 |

*5 Folds cross Validation:*

| % | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
|---|---|---|---|---|---|
| CCR | 42.86 | 100 | 100 | 85.71 | 42.86 |
| FAR | 50 | 0 | 0 | 50 | 50 |

***Bayes Minimum Risk***

*Re-substitution classification:*

Classification results:

CCR = 88.33%

FAR = 11.67%

Confusion Matrix:

| | Is Red | Is Blue |
|---|---|---|
| Classified as red | 18 | 2 |
| Classified as Blue | 2 | 13 |

*5 Folds cross Validation:*

| % | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
|---|---|---|---|---|---|
| CCR | 42.86 | 100 | 100 | 100 | 57.14 |
| FAR | 50 | 0 | 0 | 0 | 50 |

## EXPERIMENT 2

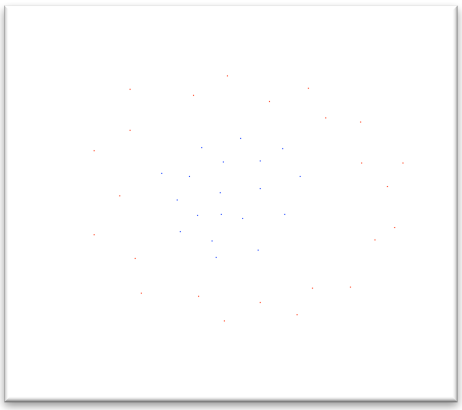


Figure A5 – Experiment 2 subject Image

Total Samples: 43 (24 red, 19 blue)

***SVM classification with dot product Kernel***

*Re-substitution classification:*

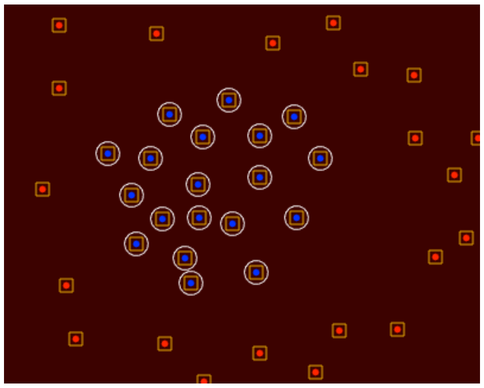


Figure A6 – Experiment 2 SVM dot product classification

Classification results:

CCR = 50%

FAR = 44.19%

Confusion Matrix:

| | Is Red | Is Blue |
|---|---|---|
| Classified as red | 24 | 19 |
| Classified as Blue | 0 | 0 |

*5 Folds cross Validation:*

| % | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
|---|---|---|---|---|---|
| CCR | 0 | 0 | 0 | 0 | 0 |
| FAR | 100 | 100 | 100 | 100 | 100 |

***SVM classification with a dot product/quadratic polynomial multi kernel0***

*Re-substitution classification:*

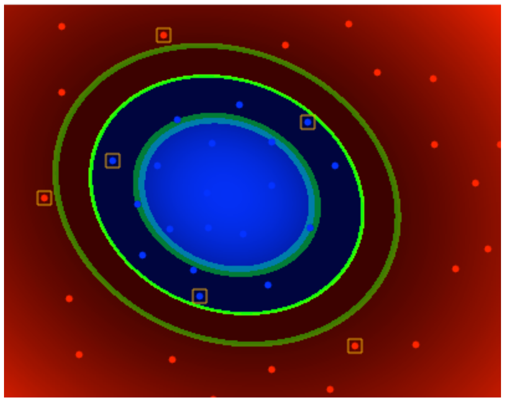


Figure A7 – Experiment 2 SVM dot product/quadratic polynomial multi kernel

Parameters used:

c=1x10^ (20)

Multi kernel weights: dot product 50%, quadratic polynomial 50%

Polynomial constant=1

Classification results:

CCR = 100%

FAR = 0%

Confusion Matrix:

| | Is Red | Is Blue |
|---|---|---|
| Classified as red | 24 | 0 |
| Classified as Blue | 0 | 19 |

*5 Folds cross Validation:*

| % | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
|---|---|---|---|---|---|
| CCR | 100 | 0 | 50 | 100 | 0 |
| FAR | 0 | 100 | 66.67 | 0 | 100 |

***SVM classification with a Gaussian radial based function kernel***

*Re-substitution classification:*

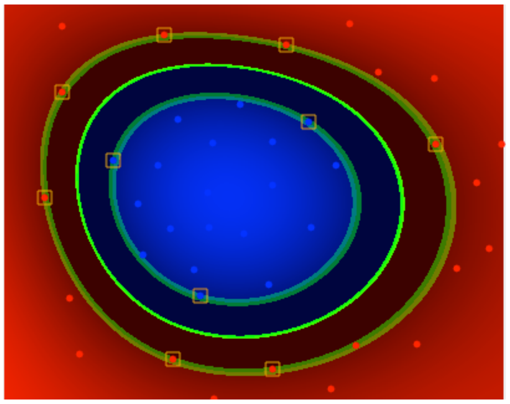


Figure A8 – Experiment 2 SVM Gaussian rbf kernel

Parameters used:
c=100
sigma=100

Classification results:
CCR = 100%
FAR = 0%

Confusion Matrix:

| | Is Red | Is Blue |
|---|---|---|
| Classified as red | 24 | 0 |
| Classified as Blue | 0 | 19 |

*5 Folds cross Validation:*

| % | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
|---|---|---|---|---|---|
| CCR | 88.89 | 100 | 100 | 100 | 100 |
| FAR | 50 | 0 | 0 | 0 | 0 |

***Bayes Minimum Risk***

*Re-substitution classification:*

<u>Classification results:</u>
CCR = 97.37%
FAR = 2%

Confusion Matrix:

| | Is Red | Is Blue |
|---|---|---|
| Classified as red | 18 | 0 |
| Classified as Blue | 1 | 24 |

*5 Folds cross Validation:*

| % | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
|---|---|---|---|---|---|
| CCR | 11 | 100 | 66.66 | 75 | 0 |
| FAR | 50 | 0 | 12.5 | 50 | 100 |

## EXPERIMENT 3

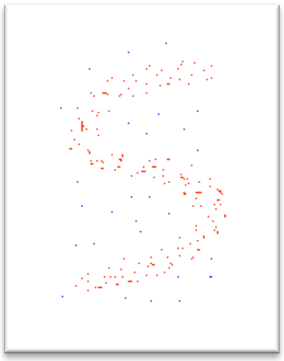


Figure A9 – Experiment 3 subject Image

Total Samples: 215 (187 red / 28 blue)

***SVM classification with dot product Kernel***
*Re-substitution classification:*

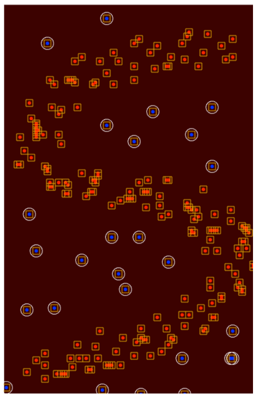


Figure A10 – Experiment 3 SVM dot product classification

<u>Classification results:</u>
CCR = 50%
FAR = 13.02%

Confusion Matrix:

| | Is Red | Is Blue |
|---|---|---|
| Classified as red | 187 | 28 |
| Classified as Blue | 0 | 0 |

*10 Folds cross Validation:*

| % | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 | Fold 6 | Fold 7 | Fold 8 | Fold 9 | Fold 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| CCR | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 50 | 0 |
| FAR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3.33 | 100 |

***SVM classification with cubic polynomial kernel***
*Re-substitution classification:*

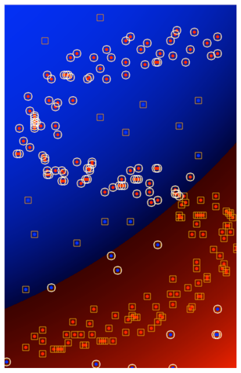


Figure A11 – Experiment 3 SVM cubic classification

Parameters used:
c=100

Classification results:
CCR = 51.57%
FAR = 49.28%

Confusion Matrix:

| | Is Red | Is Blue |
|---|---|---|
| Classified as red | 86 | 12 |
| Classified as Blue | 101 | 16 |

*10 Folds cross Validation:*

| % | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 | Fold 6 | Fold 7 | Fold 8 | Fold 9 | Fold 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| CCR | 22.73 | 18.18 | 31.82 | 40.91 | 36.36 | 38.1 | 66.67 | 80.95 | 85.71 | 52.38 |
| FAR | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 6.25 | 50 |

***SVM classification with radial base function kernel***
*Re-substitution classification:*

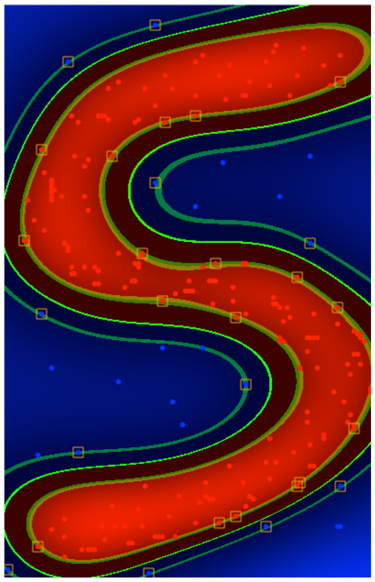


Figure A12 – Experiment 3 SVM rbf classification

Parameters used:
c=100
sigma=30

<u>Classification results:</u>
CCR = 100%
FAR = 0%

Confusion Matrix:

| | Is Red | Is Blue |
|---|---|---|
| Classified as red | 187 | 0 |
| Classified as Blue | 0 | 28 |

*10 Folds cross Validation:*

| % | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 | Fold 6 | Fold 7 | Fold 8 | Fold 9 | Fold 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| CCR | 95.45 | 100 | 100 | 100 | 100 | 100 | 100 | 95.24 | 50 | 23.81 |
| FAR | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 50 | 33.33 | 50 |

**EXPERIMENT 4**
This experiment consists on testing the SVM algorithm against a dataset with 100 features. The dataset I used for this purpose is the Hill-Valley Data Set from the UCI Machine Learning Repository. It has only two classes, 1 and 0, and the different points represent Hill heights, creating the shape of a hill when plotted sequentially.

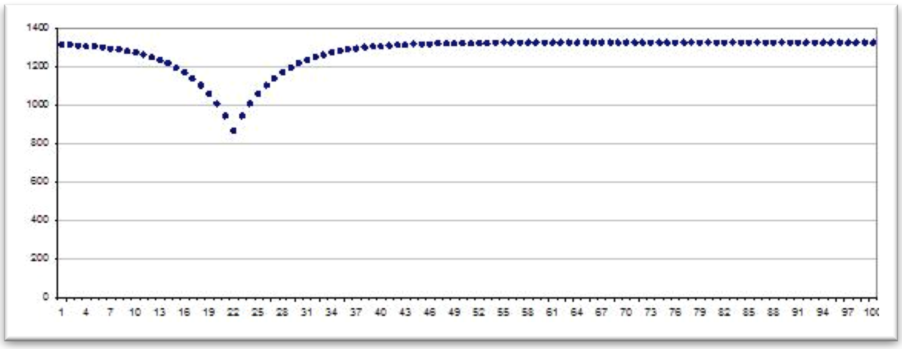


Figure A13 – Experiment 4 example.

***SVM classification with dot product Kernel***
*Re-substitution classification:*

<u>Classification results:</u>
CCR = 99.83%
FAR = 0.16%

Confusion Matrix:

| | Is class 1 | Is class 0 |
|---|---|---|
| Classified as class 1 | 300 | 0 |
| Classified as class 0 | 1 | 305 |

*10 Folds cross Validation:*

| % | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 | Fold 6 | Fold 7 | Fold 8 | Fold 9 | Fold 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| CCR | 100 | 100 | 100 | 100 | 100 | 50 | 86.49 | 100 | 100 | 100 |
| FAR | 0 | 0 | 0 | 0 | 0 | 47.54 | 15.15 | 0 | 0 | 0 |

***Bayes Minimum Risk***
*Re-substitution classification:*

Classification results:
CCR = 51.76%
FAR = 43.84%

Confusion Matrix:

| | Is class 1 | Is class 0 |
|---|---|---|
| Classified as class 1 | 29 | 18 |
| Classified as class 0 | 276 | 283 |

*10 Folds cross Validation:*

| % | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 | Fold 6 | Fold 7 | Fold 8 | Fold 9 | Fold 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| CCR | 52.8 | 52.6 | 51.45 | 56.71 | 46.61 | 47.95 | 54.05 | 52.19 | 49.78 | 51.79 |
| FAR | 42.!2 | 43.65 | 46.43 | 40.06 | 63.82 | 56.79 | 29.82 | 38.60 | 50.89 | 22.88 |