SUPPORT VECTOR CLASSIFIER VIA *MATHEMATICA*

B. PALÁNCZ[1], L. VÖLGYESI[2,3]

[1] Department of Photogrammetry and Geoinformatics
[2] Department of Geodesy and Surveying
Budapest University of Technology and Economics
[3] Physical Geodesy and Geodynamic Research Group of the Hungarian Academy of Sciences
H-1521 Budapest, Hungary
e-mail: palancz@epito.bme.hu

Abstract

In this case study a Support Vector Classifier function has been developed in *Mathematica.* Starting with a brief summary of support vector classification method, the step by step implementation of the classification algorithm in *Mathematica* is presented and explained. To check our function, two test problems, learning a chess board and classification of two intertwined spirals are solved. In addition an application to filtering of airborne digital land image by pixel classification is demonstrated using a new SVM kernel family, the KMOD, a kernel with moderate decreasing.

Keywords: software *Mathematica*, kernel methods, pixel classification, remote sensing.

Introduction

Kernel Methods are relatively new family of algorithms that presents a series of useful features for pattern analysis in datasets. Kernel Methods combine the simplicity and computational efficiency of linear algorithms, such as the perception algorithm or ridge regression, with flexibility of nonlinear systems, such as for example neural networks, and rigour of statistical approaches such as regularization methods in multivariate statistics. As a result of the special way they represent functions, these algorithms typically reduce the learning step to convex optimization problem that can always be solved in polynomial time, avoiding the problem of local minima typical of neural networks, decision trees and other nonlinear approaches [1].

1 Support Vector Classification

1.1 Binary classification

In case of binary classification, we try to estimate a real-valued function *f*: $X \subseteq R^n \to R$ using training data, that is *n* - dimensional patterns x_i and class labels $y_i \in \{-1, 1\}$

$$((x_1, y_1), ..., (x_m, y_m)) \in R^n \times \{-1, 1\}$$

such that *f* will correctly classify new examples (*x, y*) – that is, *f*(*x*) = *y* for examples (*x, y*), which were generated from the same underlying probability distribution *P*(*x,y*) as the training data. If we put no restriction on the class of functions that we choose

our estimate *f* from, however, even a function that does well on the training data – for example by satisfying $f(x_i) = y_i$ for $i = 1...m$ – need not generalize well to unseen examples. Suppose we know nothing additional about *f* (for example about its smoothness), then the values on the training patterns carry no information whatsoever about values on novel patterns. Hence learning is impossible, and minimizing the training error does not imply a small expected test error.

Statistical learning theory, or Vapnik-Chervonenkis theory, shows that is crucial to restrict the class of functions that the learning machine can implement to one with capacity that is suitable for the amount of available training data.

1.2 Optimal hyperplane classifier

To design learning algorithms, we thus must come up with a class of functions whose capacity can be computed. SV classifiers are based on the class of hyperplanes

$$\langle w, x \rangle + b = 0 \qquad w \in R^n, \qquad b \in R$$

corresponding to decision functions

$$f(x) = sign(\langle w, x \rangle + b).$$

We can show that the optimal hyperplane, defined as the one with the maximal margin of separation between the two classes (see *Fig. 1*), has the lowest capacity, which ensuring that the classifier learned from training samples will misclassify the less elements of the test samples originated from the same probability distribution.

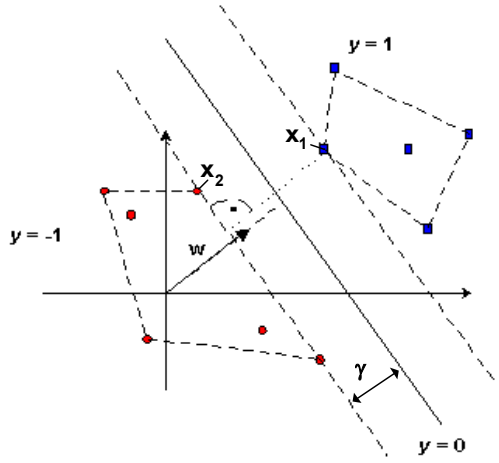

Fig. 1. A separable classification problem. The optimal hyperplane is orthogonal to the shortest line connecting the convex hulls of the two classes, and intersects it half way. There is a weight vector *w* and a threshold *b* such that $y_i(\langle w, x_i \rangle + b) > 0$. Rescaling *w* and *b* such that the point(s) closest to the hyperplane satisfy $|\langle w, x_i \rangle + b| = 1$, we obtain a form (*w, b*) of the hyperplane with $y_i(\langle w, x_i \rangle + b) \geq 1$. Note that the margin, measured perpendicularly to the hyperplane, equals $1/\|w\|$. To maximize the margin, we thus have to minimize $\|w\|$ subject to $y_i(\langle w, x_i \rangle + b) \geq 1$ [2].

1.3 Maximal margin classifier

The optimization problem to find the optimal w vector and the threshold b is the following, given a set of linearly separable training samples

$$S = ((x_1, y_1), ..., (x_m, y_m))$$

the hyperplane (w^*, b^*) that maximizes the geometric margin.

$$minimize_{w,b} \langle w, w \rangle$$

$$subject\ to\ y_i \langle w, x_i \rangle + b \geq 1, \quad i = 1, ... m\ .$$

Then the geometric margin can be computed considering that

$$\langle w^*, x_1 \rangle + b^* = 1$$
$$\langle w^*, x_2 \rangle + b^* = -1$$

then

$$\langle w^*, (x_1 - x_2) \rangle = 2$$

rescaling

$$\left\langle \frac{w^*}{\|w^*\|}, (x_1 - x_2) \right\rangle = \frac{2}{\|w^*\|}$$

therefore the margin is

$$\gamma = \frac{1}{\|w^*\|}\ .$$

The training patterns lie closest to the hyperplane (see *Fig. 1* two balls and one diamond) are called support vectors, carrying all relevant information about the classification problem. The number of support vectors, *SV* are equal or less than the number of the training patterns, *m*.

This minimization problem can be transformed into a dual maximization problem leading to a quadratic programming task, whose solution w has an expansion

$$w = \sum_{i=1}^{SV} v_i x_i$$

Consequently, the final decision function is

$$f(x) = sign\left(\sum_{i=1}^{SV} v_i \langle x, x_i \rangle + b \right)$$

which depends only on dot products between patterns. This lets us generalize to the nonlinear case.

1.4 Feature spaces and kernels

The *Fig. 2* shows the basic idea of SV machines, which is to map the data into some other dot space, called the feature space F via nonlinear map,

$$\Phi : R^n \rightarrow F$$

and perform the above linear algorithm in *F*. This is only requires the evaluation of dot products,

$$K(u,v) = \langle \Phi(u), \Phi(v) \rangle$$

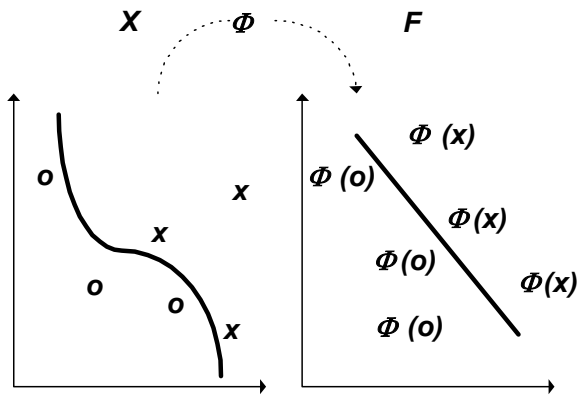

Fig. 2. The idea of SV machines: map the training data nonlinearly into a higher dimensional feature space via Φ, and construct a separating hyperplane with maximum margin there. This yields a nonlinear decision boundary in input space. By the use of kernel function, it is possible to compute the separating hyperplane without explicitly carrying out the map into the feature space [3].

Clearly, if *F* is high dimensional, the dot product on the right hand side will be very expensive to compute. In some cases, however there is a simple kernel that can be evaluated efficiently. For instance, the polynomial kernel

$$K(u,v) = \langle u,v \rangle^d$$

can be shown to correspond to a map Φ into the space spanned by all products of exactly *d* dimensions of R^n. For $d = 2$ and $u, v \in R^2$, for example, we have

$$\langle u,v \rangle^2 = \left(\left\langle \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}, \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \right\rangle \right)^2 = \left\langle \begin{pmatrix} u_1^2 \\ \sqrt{2}\, u_1 u_2 \\ u_2^2 \end{pmatrix}, \begin{pmatrix} v_1^2 \\ \sqrt{2}\, v_1 v_2 \\ v_2^2 \end{pmatrix} \right\rangle = \langle \Phi(u), \Phi(v) \rangle$$

defining $\Phi(x) = (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$.

More generally, we can prove that for every kernel that gives rise to a positive matrix (kernel matrix) $M_{ij} = K(x_i, x_j)$, we can construct a map such that $K(u,v) = \langle \Phi(u), \Phi(v) \rangle$ holds.

1.5 Optimization as a dual quadratic programming problem

Now the dual minimization problem of margin maximization is the following, consider classifying a set of training samples,

$$S = ((x_1, y_1), \ldots (x_m, y_m))$$

using the feature space implicitly defined by the kernel $K(x,z)$ and suppose the parameters α^* solve the following quadratic optimization problem,

$$\textit{minimize} \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \left(K(x_i, x_j) + \frac{1}{c} \delta_{ij} \right)$$

$$\textit{subject to} \quad \sum_{i=1}^{m} y_i \alpha_i = 0, \quad \alpha_i \geq 0, \quad i = 1, \ldots m .$$

Let $f(x) = \sum_{i=1}^{m} y_i \alpha_i^* K(x_i, x) + b^*$, where b^* is chosen so that $y_i \ f(x_i) = 1 - \frac{\alpha_i^*}{c}$ for any i with $\alpha_i^* \neq 0$.

Then the decision rule given by $sign(f(x))$ is equivalent to the hyperplane in the feature space implicitly defined by the kernel $K(x,z)$, which solves the optimization problem, where the geometric margin is

$$\gamma = \left(\sum_{i \in sv} \alpha_i^* - \frac{1}{c} \left\langle \alpha^*, \alpha^* \right\rangle \right)^{-1/2}$$

where set *sv* corresponds to indexes *i*, for which $\alpha_i^* \neq 0$,

$$sv = \left\{ i : \quad \alpha_i^* \neq 0; \quad i = 1, \ldots m \right\}$$

Training samples, x_i for which $i \in sv$ are called support vectors giving contribution to the definition of $f(x)$.

2 Implementation of SVC in *Mathematica*

2.1 Steps of implementation

The dual optimization problem can be solved conveniently using *Mathematica*. In this section, the steps of the implementation of SVC algorithm are shown by solving XOR problem. The truth table of XOR, using bipolar values for the output, is

Table 1. Truth table of XOR problem

x1	x2	y
0	0	-1
0	1	1
1	0	1
1	1	-1

The input and output data lists are

```
xym={{0,0},{0,1},{1,0},{1,1}};
zm={-1,1,1,-1};
```

Let us employ Gaussian kernel with β gain

```
β=10.;
```

```
K[u_,v_]:=Exp[-β (u-v).(u-v)]
```

The number of the data pairs in the training set, *m* is

```
m=Length[zm]
```

```
4
```

Create the objective function $W(\alpha)$ to be maximized, with regularization parameter,

```
c=5.;
```

First, we prepare a matrix *M,* which is an extended form of the kernel matrix,

```
M=(Table[N[K[xym[[i]],xym[[j]]]],
   {i,1,m},{j,1,m}]+(1/c)IdentityMatrix[m]);
```

then the objective function can be expressed as,

```
W=∑(i=1 to m) αi -(1/2) ∑(i=1 to m) ∑(j=1 to m) (zm[[i]]zm[[j]] αi αj M[[i,j]]);
```

The constrains for the unknown variables are

```
g=Apply[And,Join[Table[αi ≥0,{i,1,m}],{∑(i=1 to m) zm[[i]]αi==0}]];
```

However the maximization problem is a convex quadratic problem, from practical reasons to maximize the objective function the built in function `NMaximize` is applied. `NMaximize` implements several algorithms for finding constrained global optima. The methods are flexible enough to cope with functions that are not differentiable or continuous, and are not easily trapped by local optima. Possible settings for the `Method` option include `"RandomSearch"`, `"NelderMead"`, `"DifferentialEvolution"` and `"SimulatedAnnealing"`.

Here we use `DifferentialEvolution,` which is a genetic algorithm that maintains a population of specimens, $x_1, ..., x_n$, represented as vectors of real numbers ("genes"). Every iteration, each x_i chooses random integers *a*, *b*, and *c* and constructs the mate $y_i = x_i + \gamma(x_a + (x_b - x_c))$, where γ is the value of `ScalingFactor`. Then x_i is mated with y_i according to the value of `CrossProbability`, giving us the child z_i. At this point x_i competes against z_i for the position of x_i in the population. The default value of `SearchPoints` is `Automatic`, which is `Min[10*d, 50]`, where *d* is the number of variables.

We need the list of unknown variables α,

```
vars=Table[αi,{i,1,m}];
```

Then the solution of the maximization problem is,

```
sol=NMaximize[{W,g},vars,Method→DifferentialEvolution]
{1.66679,{α1→0.833396, α2→0.833396, α3→0.833396, α4→0.833396}}
```

The consistency of this solution can be checked by computing values of *b* for every data points. Theoretically, these values should be same for any data points, however, in general, this is only approximately true.

```
bdata=Table[((1-αj/c)/zm[[j]]-
      ∑(i=1 to m) zm[[i]]αi K[xym[[i]],xym[[j]]])/.sol[[2]],{j,1,m}]
```

```
{-1.89729×10^-16, 6.65294×10^-17, 3.45251×10^-16, 0.}
```

The value of *b* can be chosen as the average of these values

```
b=Apply[Plus,bdata]/m
5.55126×10^-17
```

Then the classifier function is,

```
f[w_]:=(( Σ_{i=1}^{m} zm[[i]]α_i K[w,xym[[i]]])+b)/.sol[[2]]
```

In symbolic form

```
Clear[x,y]
f[{x,y}]
5.55126×10^-17-0.833396 e^-10.((-1+x)^2+(-1+y)^2) +0.833396 e^-10.(x^2+(-1+y)^2)
             +0.833396 e^-10.((-1+x)^2+y^2) -0.833396 e^-10.(x^2+y^2)
```

Let us display the contour lines of the continuous classification function,

```
<<ExtendGraphics`LabelContour`
p=ContourPlot[f[{x,y}], {x, 0, 1},{y,0,1},
  Contours→{-0.5,-0.05,0,0.05,0.5},ContourShading→False,
  PlotPoints→50,DisplayFunction→Identity];
p1=LabelContourLines[p,LabelPlacement→Automatic,
  LabelFont→{"Courier",10},DisplayFunction→Identity];
<<Graphics`MultipleListPlot`
p2=MultipleListPlot[{{0,0},{1,1}},{{0,1},{1,0}},
  SymbolStyle→{Hue[.0],Hue[0.8]},SymbolShape→{PlotSymbol[Box,3],
  PlotSymbol[Star,7]},Frame→True,Axes→False,AspectRatio→1,
  DisplayFunction→Identity];
Show[{p1,p2},DisplayFunction→$DisplayFunction];
```

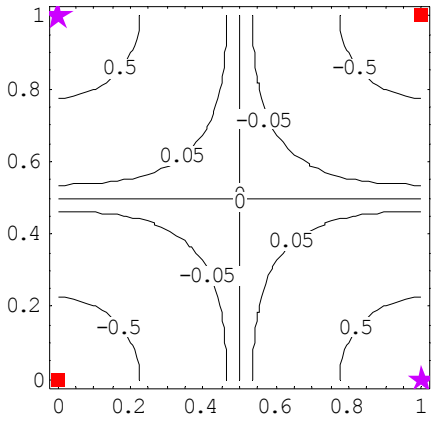

Fig. 3. The contour lines of the continuous classification function *f(x*1*, x*2*)* for XOR problem

The discrete classifier, the decision rule using signum function is

```
Plot3D[Sign[f[{x,y}]],{x,0,1},{y,0,1},PlotRange→All];
```

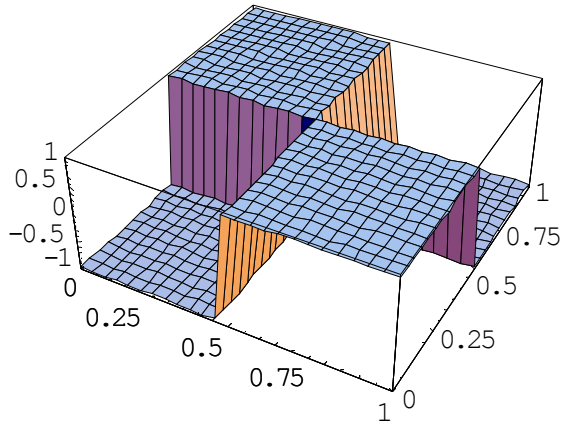

Fig. 4. The decision rule, sign($f(x1, x2)$) for XOR problem

```
Map[Sign[f[#]]&,xym]
{-1,1,1,-1}
```

and

```
zm==%
True
```

2.2 Mathematica modul for SVC

These steps can be collected in a module, where the vector *xm* contains the input vectors (the training set), and the vector *ym* contains the corresponding scalar output values (the labels of the training set),

```
SupportVectorClassifier[xm_,ym_,K_,c_]:=Module[
    {m,n,M,i,j,W,g,vars,sol,bdata,b},
    m=Length[ym];n=Length[xm[[1]]];
    M=Table[K[xm[[i]],xm[[j]]],{i,1,m},{j,1,m}]+(1/c)
    IdentityMatrix[m];
    W=∑_{i=1}^{m} αi -(1/2) ∑_{i=1}^{m} ∑_{j=1}^{m} (ym[[i]]ym[[j]] αi αj M[[i,j]]);
    g=Apply[And,Join[Table[αi ≥0,{i,1,m}],{∑_{i=1}^{m} ym[[i]]αi==0}]];
    vars=Table[αi,{i,1,m}];
    sol=NMaximize[{W,g},vars,Method→DifferentialEvolution][[2]];
    bdata=Table[((1-αj/c)/ym[[j]]-
          ∑_{i=1}^{m} ym[[i]]αi K[xm[[i]],xm[[j]]])/.sol[[2]],{j,1,m}];
    b=Apply[Plus,bdata]/m;
    {((∑_{i=1}^{m} ym[[i]]αi K[Table[xj,{j,1,n}],xm[[i]]])+b)
       /.sol,vars/.sol}];
```

The results of this module are the analytic form of the continuous classifier function and the values of α_i's. Let us check the solution of the XOR problem

```
SupportVectorClassifier[xym,zm,K,c]
```

```
{5.55112×10^-17-0.833396 e^-10.((-1+x1)^2+(-1+x2)^2) +0.833396 e^-10.(x1^2+(-1+x2)^2)
   +0.833396 e^-10.((-1+x1)^2+x2^2) -0.833396 e^-10.(x1^2+x2^2),
   {0.833396, 0.833396, 0.833396, 0.833396}}
```

```
%[[1]]==f[{x1 , x2 }]//Chop
True
```

3 Two test problems

3.1 Learning a chess board

Let us consider a 2 × 2 chess board. The training points are generated by uniformly distributed random numbers from the interval [-1,1] × [-1, 1]. The chess board matrix

```
M={{1,-1},{-1,1}};
p1=ListDensityPlot[M,MeshRange→{{-1,1},{-1,1}},
   DisplayFunction→Identity];
```

Creating the training set using 50 random samples,

```
xym={};zm={};
Do[x1=Random[Real,{-0.99,0.99}];x2=Random[Real,{-0.99,0.99}];
  If[x1 x2>0,z=1,z=-1];
    AppendTo[xym,{x1,x2}];AppendTo[zm,z],{k,1,50}];
```

Preparation data to display them

```
data=Transpose[Join[Transpose[xym],{zm}]];
data1=Map[Drop[#,-1]&,Select[data,#[[3]]>0&]];
data2=Map[Drop[#,-1]&,Select[data,#[[3]]<0&]];
p2=MultipleListPlot[data1,data2,
  SymbolShape→{PlotSymbol[Triangle,5],
  PlotSymbol[Box,2]},SymbolStyle→{Hue[.7],Hue[.0]},Frame→True,
  AspectRatio→1,DisplayFunction→Identity];
```

Let us employ the same Gaussian kernel, but now with gain $\beta = 15$.

```
β=15;
```

employing parameter c = 100,

```
c=100;
```

The solution is

```
F=SupportVectorClassifier[xym,zm,K,c];
```

This run could take some minutes.

```
p3= ContourPlot[F[[1]],{x1,-1,1},{x2,-1,1},Contours→{0},
   PlotPoints→100,ContourShading→False,DisplayFunction→Identity];
p4=DensityPlot[Sign[F[[1]]],{x1,-1,1},{x2,-1,1},
   PlotPoints→100,Mesh→False,DisplayFunction→Identity];
Show[GraphicsArray[{Show[{p2,p3}],p1,p4}],
   DisplayFunction→$DisplayFunction];
```

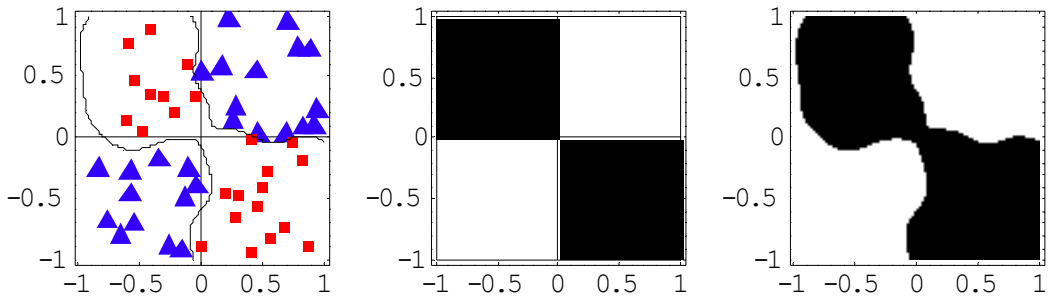

Fig. 5. SVC result from 50 random samples: the functions *f(x)* with the random samples, the ideal chess board and sign(*f*(*x*))

The SV solution of a larger chess board problem can be found in [3].

3.2 Two intertwined spirals

The two intertwined spirals is a challenging classification benchmark originated from the field of neural networks [4].

Parametric equations of the spirals are

```
x1[t_]:=2 Cos[t] e^(t/10)
y1[t_]:=1.5 Sin[t] e^(t/10)
x2[t_]:=2.7 Cos[t] e^(t/10)
y2[t_]:=2.025 Sin[t] e^(t/10)
```

Generating 26 discrete points for each spiral,

```
s1=Table[{x1[t],y1[t]},{t, π, 3.5π, 2.5π/25}];
s2=Table[{x2[t],y2[t]},{t, -π/2, 2.5π, 3.0π/25}];
```

then displaying these points,

```
S1=ListPlot[s1,PlotStyle→{RGBColor[1,0,0],PointSize[0.02]},
   AspectRatio→1,DisplayFunction→Identity];
S2=ListPlot[s2,PlotStyle→{RGBColor[0,0,1],PointSize[0.015]},
   AspectRatio→1,DisplayFunction→Identity];
pspiral=Show[{S1,S2},DisplayFunction→$DisplayFunction];
```

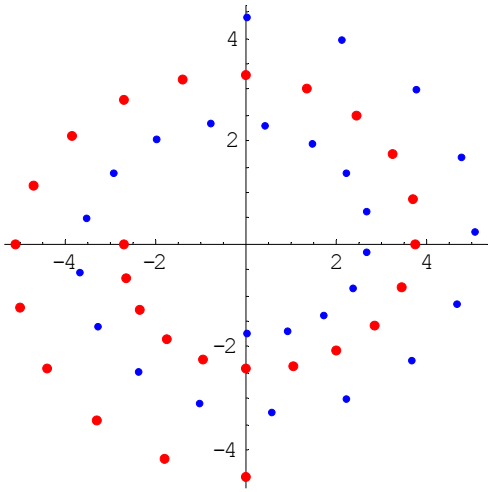

Fig. 6. Two intertwined spirals represented by 26 points each

Creating the teaching set, putting these points into one list

```
xym=Join[s1,s2];
```

Generating the labels of the samples,

```
Generating the labels of the samples,
zm=Join[Table[1.,{26}],Table[-1.,{26}]];
Length[zm]
52
Dimensions[xym]
{52,2}
```

Applying wavelet kernel [5] with parameter $a = 1.8$, in case of dimension $n = 2$

```
n=2;a=1.8;
```

$$K[u\_,v\_]:=\prod_{i=1}^{n}\left(\cos[1.75(u[[i]]-v[[i]])/a]\ \exp[-(u[[i]]-v[[i]])^2/2a^2]\right)$$

and with parameter $c = 100$

```
c=100.;
```

The solution is

```
F=SupportVectorClassifier[xym,zm,K,c];
```

This run could take some minutes.

```
psvc1=ContourPlot[F[[1]],{x1,-7,7},{x2,-7,7},Contours→{0},
      PlotPoints→50,AspectRatio→1,ContourShading→False,
      DisplayFunction→Identity];
Show[{pspiral,psvc1},DisplayFunction→$DisplayFunction];
```

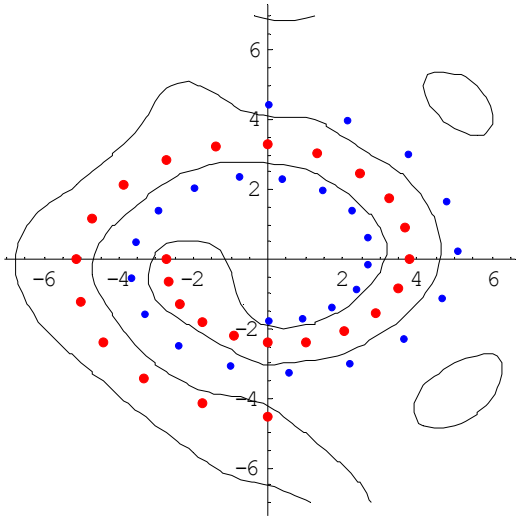

Fig. 7. Classification results of SVC with the nonlinear decision boundary

The continuous classification function and the decision rule can be displayed in 3D, too

```
P1=Plot3D[F[[1]],{x1,-7,7},{x2,-7,7},PlotRange→All,
   PlotPoints→{30,30},DisplayFunction→Identity];

P2=Plot3D[Sign[F[[1]]],{x1,-7,7},{x2,-7,7},PlotRange→All,
```

```
   PlotPoints→{40,40},DisplayFunction→Identity];
Show[GraphicsArray[{P1,P2}],DisplayFunction→$DisplayFunction];
```

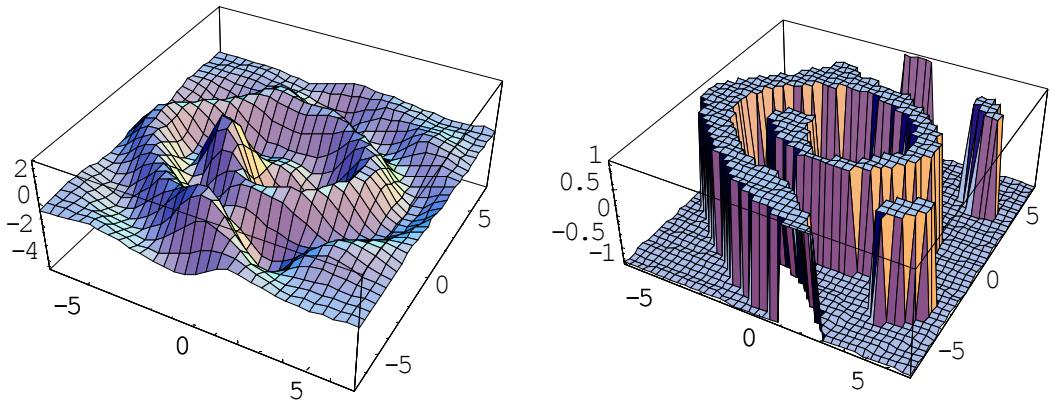

Fig. 8. Classification function and the corresponding decision rule

4 Image classification

Classification of digital images in order to separate different categories of land cover types, like urban area, water, vegetation, agricultural area etc. and to carry out thematic change analysis is a frequent task in geoscience and remote sensing. Many different methods are used, traditional maximum-likelihood, *k*- nearest neighbor, rule based (classification and regression tree), supervised and unsupervised neural network, fuzzy logic and neuro-fuzzy and even support vector classifiers with Gaussian -kernel.

SVC were used for classification for land cover using polarimetric synthetic aperture radar (SAR) images, and for classification for clouds, snow and ice, however with usual kernels, like RBF (Radial Basis Function). In this illustrative example, we use KMOD type kernel for synthetic image data.

Let us load the Image Processing Application of *Mathematica*,

```
<<ImageProcessing`
img=ImageRead["F:\ImageProcessing\Syntetic.JPG"];
```

Let us consider a relatively small image of synthetic data,

```
ImageDimensions[img]
{201,301}
Show[Graphics[img]];
```

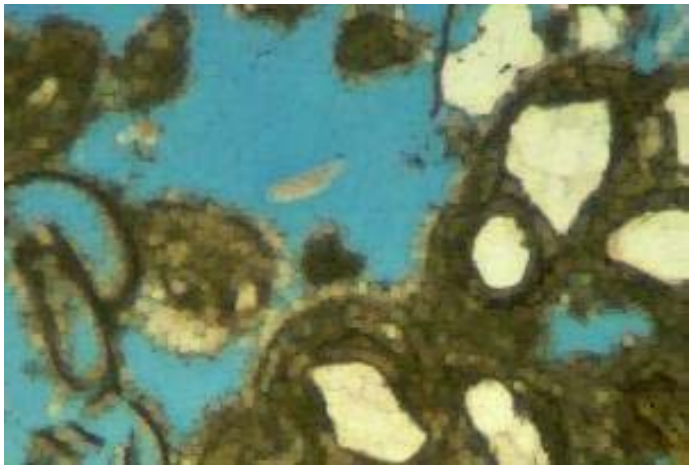

Fig. 9. Digital image of synthetic data

4.1 Binary classification

We should like to filter out pixels being different from yellowy type ones. For this task an SVM binary classifier can be developed. Pixels from the three different categories: yellowy, brownish and bluish spots are picked up randomly and their RGB values are stored in three different files.

We have ten yellowish pixels,

```
rgb1=ReadList["F:\ImageProcessing\yellow.dat",{Number,Number,Number}];
Show[Graphics[RasterArray[{RGBColor[#[[1]],#[[2]],#[[3]]]&/@rgb1}]]];
```

Fig. 10. Colors (grayscale) of the ten yellowish training data

ten brown pixels,

```
rgb2=ReadList["F:\ImageProcessing\dark.dat",{Number,Number,Number}];
Show[Graphics[RasterArray[{RGBColor[#[[1]],#[[2]],#[[3]]]&/@rgb2}]]];
```

Fig. 11. Colors (grayscale) of the ten brownish traning data

and ten bluish pixels,

```
rgb3=ReadList["F:\ImageProcessing\light.dat",{Number,Number,Number}];
Show[Graphics[RasterArray[{RGBColor[#[[1]],#[[2]],#[[3]]]&/@rgb3}]]];
```

Fig. 12. Colors (grayscale) of the ten bluish training data

These three sets are joined building the input for the classifier,

```
xym=Join[rgb1,rgb2,rgb3];
```

The first ten elements are labeled with 1, and the remaining elements with -1,

```
zm=Join[Table[1,{10}],Table[-1,{20}]];
```

Now, we shall employ a Kernel with MOderate Decreasing (KMOD) having two parameters,

```
γ=0.5;σ=3.;
```

The motivation of the employment of this kernel can be explained as it follows. In most commonly used kernels (eg. RBF), points very close to each other are strongly correlated whereas points far apart have uncorrelated images in the augmented space. The aim is, to force the images of the original points to be linearly separable in the augmented space. In order to get such a behavior, a kernel must turn very close points from the original space into weakly correlated elements (as weak as possible) while still maintaining the closeness information from vanishing. To achieve this tradeoff, we need the following couple features: a quick decrease in the neighborhood of zero and a moderate decrease towards infinity. The RBF kernel may satisfy correctly the first requirement but not the second, whereas the exponential RBF does not respond correctly for both of the requirements. Alternatively, the KMOD is proposed [7], whose analytic expression is,

```
K[u_,v_]:=Exp[γ/(Norm[u-v]^2+σ^2)]-1
```

In case of n =1

```
Plot3D[ Exp[γ/(Abs[u-v]^2+σ^2)]-1,{u,0,1},{v,0,1}];
```

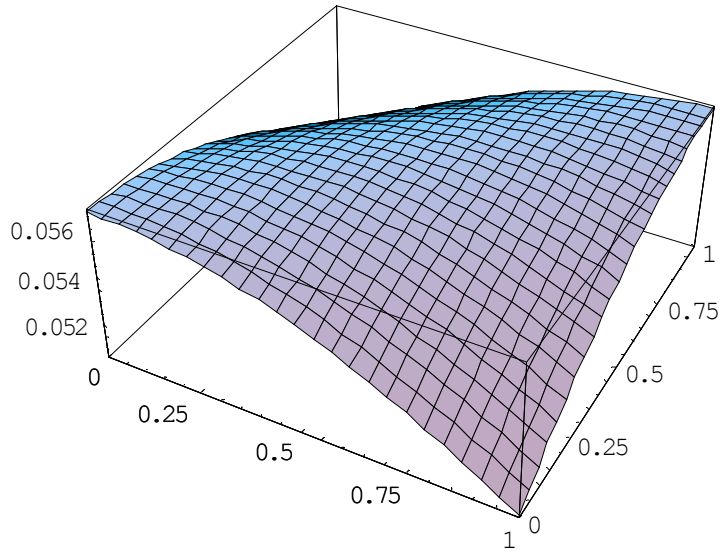

Fig. 13. Kernel with MOderate Decreasing (KMOD)

The value of the regularization parameter is,

```
c=100;
```

Training the continuous classifier function,

```
F1=SupportVectorClassifier[xym,zm,K,c];
```

we get its analytical form,

```
Short[F1[[1]],15]
-0.108208-91.611(-1+e^(0.5/(9.+Abs[-0.352941+x1]^2+Abs[-0.639216+x2]^2+Abs[-0.596078+x3]^2)))-
    -43.766(-1+e^(0.5/(9.+Abs[-0.34902+x1]^2+Abs[-0.627451+x2]^2+Abs[-0.596078+x3]^2)))+
    +<<30>>+
    +0.0(-1+e^(0.5/(9.+Abs[-0.25098+x1]^2+Abs[-0.262745+x2]^2+Abs[-0.219608+x3]^2)))+
    +0.0(-1+e^(0.5/(9.+Abs[-0.235294+x1]^2+Abs[-0.247059+x2]^2+Abs[-0.211765+x3]^2)))
```

The RGB values of the original image vector

```
xx=Flatten[RawImageData[img],1];
```

and its dimensions

```
nd=Dimensions[xx]
{60501,3}
```

Now, the discrete classifier is applied,

```
decision=Map[Sign[F1[[1]]]/.{x1→#[[1]],x2→#[[2]],x3→#[[3]]}&,xx/255];
```

We shall use colors for yellowish and not yellowish spots, with RGB values `pathes` and `others`, respectively,

```
patches={0.5,0.5,0};others={1,0.5,0.75};
```

The RGB value of pixels classified as yellowish (labeled with 1 by the classifier) will be overwritten by the RGB values of `pathes`, and the others pixels (labeled with –1) by the RGB values of `others`,

```
Do[If[decision[[i]]==1,xx[[i]]=patches,xx[[i]]=others],
  {i,1,nd[[1]]}];
```

We partition the RGB image vector to form a matrix, and transform this image matrix into an image object,

```
U=ToRGBColor[Partition[xx,301]];
```

Here are the original and the filtered image,

```
Show[GraphicsArray[{Graphics[img],Graphics[U]}]];
```

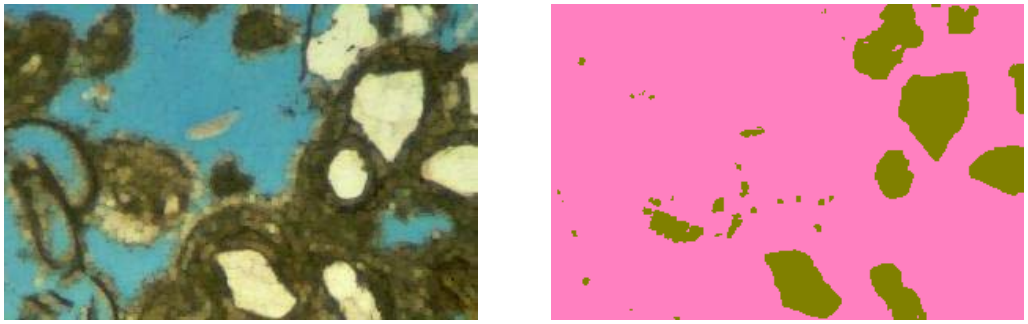

Fig. 14. The original and the filtered image

The results shows the data generalization ability of SVC, which was trained with only 30 pixels, and successfully represents more than 60 thousands.

4.2 Multi-class classification

Support Vector Classifiers were originally designed for binary classification. How to effectively extend it for multi-class classification is still an on-going research issue. Currently there are two types of approaches for multi-class SVC. One is by constructing and combining several binary classifiers, while the other is by directly considering all data in one optimization formulation. Methods of this latest category are called all-together methods.

Here we employ the *one – against – all* method [8] belonging to the first category, the combination of several binary classifiers. It constructs *k* SVC models, which means *k* decision functions, where *k* is the number of classes. The *i*th SVC is trained with all of the examples in the *i*th class with positive labels, and all other examples with negative labels. Thus given *m* training data $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in R^n, \quad i = 1, \ldots m,$ and $y_i \in \{1, k\}$. We say x_i is in the class which has the largest value of the decision function. In our case *k* = 3, *m* = 201 × 301 = 60501 and *n* = 3.

Let us define the list of labels for the second class,

```
zm=Join[Table[-1,{10}],Table[1,{10}],Table[-1,{10}]];
```

The second decision function is

```
F2=SupportVectorClassifier[xym,zm,K,c];
```

in symbolic form

```
Short[F2[[1]],15]
```

```
-0.0635132-60.3727(-1+e^(0.5/(9.+Abs[-0.352941+x1]^2+Abs[-0.639216+x2]^2+Abs[-0.596078+x3]^2)))-
   -32.4338(-1+e^(0.5/(9.+Abs[-0.34902+x1]^2+Abs[-0.627451+x2]^2+Abs[-0.596078+x3]^2)))+
   +<<27>>+
   +42.0541(-1+e^(0.5/(9.+Abs[-0.25098+x1]^2+Abs[-0.262745+x2]^2+Abs[-0.219608+x3]^2)))+
   +37.0146(-1+e^(0.5/(9.+Abs[-0.235294+x1]^2+Abs[-0.247059+x2]^2+Abs[-0.211765+x3]^2)))
```

The third class has the list of labels as follows

```
zm=Join[Table[-1,{20}],Table[1,{10}]];
```

Then the third decision function is

```
F3=SupportVectorClassifier[xym,zm,K,c];
```

in symbolic form

```
Short[F3[[1]],15]
```

```
-0.874528+157635(-1+e^(0.5/(9.+Abs[-0.352941+x1]^2+Abs[-0.639216+x2]^2+Abs[-0.596078+x3]^2)))+
   +78.926(-1+e^(0.5/(9.+Abs[-0.34902+x1]^2+Abs[-0.627451+x2]^2+Abs[-0.596078+x3]^2)))-
   -51.2964(-1+e^(0.5/(9.+Abs[-0.811765+<<1>>]^2+Abs[<<1>>]^2+Abs[-0.588235+x3]^2)))+
   +81.8819(-1+e^(0.5/(9.+<<1>>^2+<<1>>^2+Abs[<<1>>]^2)))
   +<<32>>
```

Now we restore the RGB values of the original image vector, which was overwritten during the binary classification,

```
xx=Flatten[RawImageData[img],1];
```

Then applying the three different continuous classifiers to the pixel vector of the image

```
decision1=Map[F1[[1]]/.{x1→#[[1]],x2→#[[2]],x3→#[[3]]}&,xx/255];
decision2=Map[F2[[1]]/.{x1→#[[1]],x2→#[[2]],x3→#[[3]]}&,xx/255];
decision3=Map[F3[[1]]/.{x1→#[[1]],x2→#[[2]],x3→#[[3]]}&,xx/255];
```

We construct a list having elements as list referring to a pixel, and containing three values, the values resulted by the three different classifiers applied to the pixels,

```
d123=Transpose[{decision1,decision2,decision3}];
```

The pixel will be assigned to the class, for which it has the largest value of the decision functions,

```
d=Map[Position[#,Max[#]]&,d123]//Flatten;
```

Let us assign the following three colors to the different classes,

```
class1={1,0,0};class2={0,1,0};class3={0,0,1};
```

We paint each pixel with the color of the proper class,

```
Do[Which[d[[i]]==1,xx[[i]]=class1,d[[i]]==2,xx[[i]]=class2,
  d[[i]]==3,xx[[i]]=class3],{i,1,nd[[1]]}];
```

then partition the RGB image vector to form an image matrix, and transform this image matrix into an image object,

```
U=ToRGBColor[Partition[xx,301]];
```

Here are the original and the three-class classified image,

```
Show[GraphicsArray[{Graphics[img],Graphics[U]}]];
```

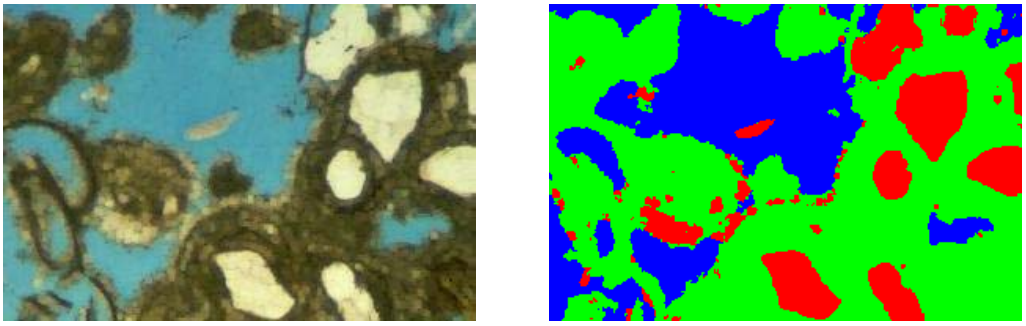

Fig. 15. The original and the three-class classified image

5 Conclusions

Support vector classification method has been developed in software *Mathematica* and the method is ready to use for different technical applications. The step by step implementation of the support vector classification algorithm in *Mathematica* was presented and explained here.

Support vector classification method provides a very promising application possibility in photogrammetry and petrological microscopy. One of the most important applications of this method in remote sensing is the filtering of airborne digital land images by pixel classification.

The *Mathematica* notebook form of this paper is available on Web [9].

Acknowledgements

Our investigations are supported by the National Scientific Research Found (OTKA T-037929). The authors wish to thank professor D. Holnapy for his valuable comments.

References

[1] Berthold M, Hand D J /Eds./ (2003): Intelligent Data Analysis, An Introduction. *Springer Verlag*.
[2] Hearst M A (1998): Support Vector Machines*, IEEE Intelligent Systems*, pp. 18 -28.
[3] Cristianini N, Shawe-Taylor J (2003): An introduction to Support Vector Machines and other kernel - based learning methods. *Cambridge, University Press*.
[4] Juillé H, Pollack J B (1996): Co-evolving intertwined spirals, *in Proc. of the 5th. Ann.Conf. on Evolutionary Programming, San Diego, USA*, pp.461-468.
[5] Zhang L, Zhou W, Jiao L (2004): Wavelet Support Vector Machine, *IEEE Trans. Systems, Man and Cybernetics - Part B: Cybernetics,* Vol. 4. No.1. pp. 34 -39, Febr. 2004.
[6] Wavelet Explorer with *Mathematica*, *Mathematica* Application Package, Wolfram Research Inc. 2003.
[7] Remaki L, Cheriet M (2000): Kcs-new kernel family with compact support scale space. *IEEE Transactions On Image Processing,* 9 (6): 970, June 2000.
[8] Bottou L, et al (1994): Comparison of classifier methods: a case study in handwriting digit recognition. In Int. *Conf. on Pattern Recognition, IEEE Computer Society Press,* pp. 77-87.
[9] Paláncz B: Support Vector Classifier, e-publication, *Wolfram Research Inc., Mathematica Information Center,* http://library.wolfram.com/infocenter/MathSource/5293/

* * *

Paláncz B, Völgyesi L. (2004) Support Vector Classifier via Mathematica. Periodica Polytechnica Civ. Eng, Vol. 48, Nr. 1-2. pp. 15-37.

Dr. Lajos VÖLGYESI, Department of Geodesy and Surveying, Budapest University of Technology and Economics, H-1521 Budapest, Hungary, Műegyetem rkp. 3.
Web: http://sci.fgt.bme.hu/volgyesi E-mail: volgyesi@eik.bme.hu