SUPPORT VECTOR MACHINE

https://docs.opencv.org/3.4/d1/d73/tutorial_introduction_to_svm.html

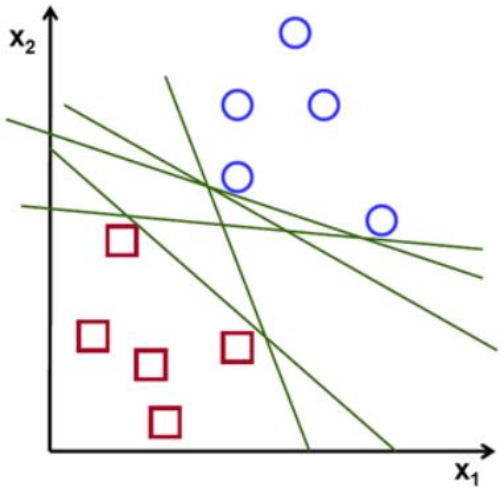
SUPPORT VECTOR MACHINE

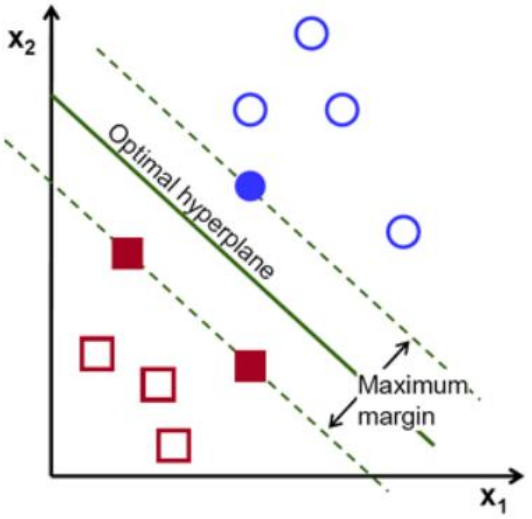
- Thuật toán SVM tìm siêu phẳng hyperplane tối ưu phân chia hai lớp.
- Siêu phẳng có phương trình $f(x) = \beta_0 + \beta^T x$

β là trọng số và β_0 là bias.

- Khoảng cách từ một điểm **x** đến siêu phẳng là

$$\text{distance} = \frac{|\beta_0 + \beta^T x|}{\|\beta\|}$$





SUPPORT VECTOR MACHINE

- Siêu phẳng tối ưu là siêu phẳng phân hai lớp sao cho khoảng cách tối thiểu margin đến hai lớp là bằng nhau.
- Siêu phẳng tối ưu được chọn với $|\beta_0 + \beta^T x| = 1$

sao cho khoảng cách đến lớp là cực đại, các điểm gần siêu phẳng tối ưu nhất gọi là support vectors, lúc đó siêu phẳng tối ưu gọi là siêu phẳng chính tắc. Với support vector $\beta_0 + \boldsymbol{\beta}^T\mathbf{x} = \pm 1$

$$\text{distance}_{\text{support vectors}} = \frac{|\beta_0 + \beta^T x|}{\|\beta\|} = \frac{1}{\|\beta\|}$$

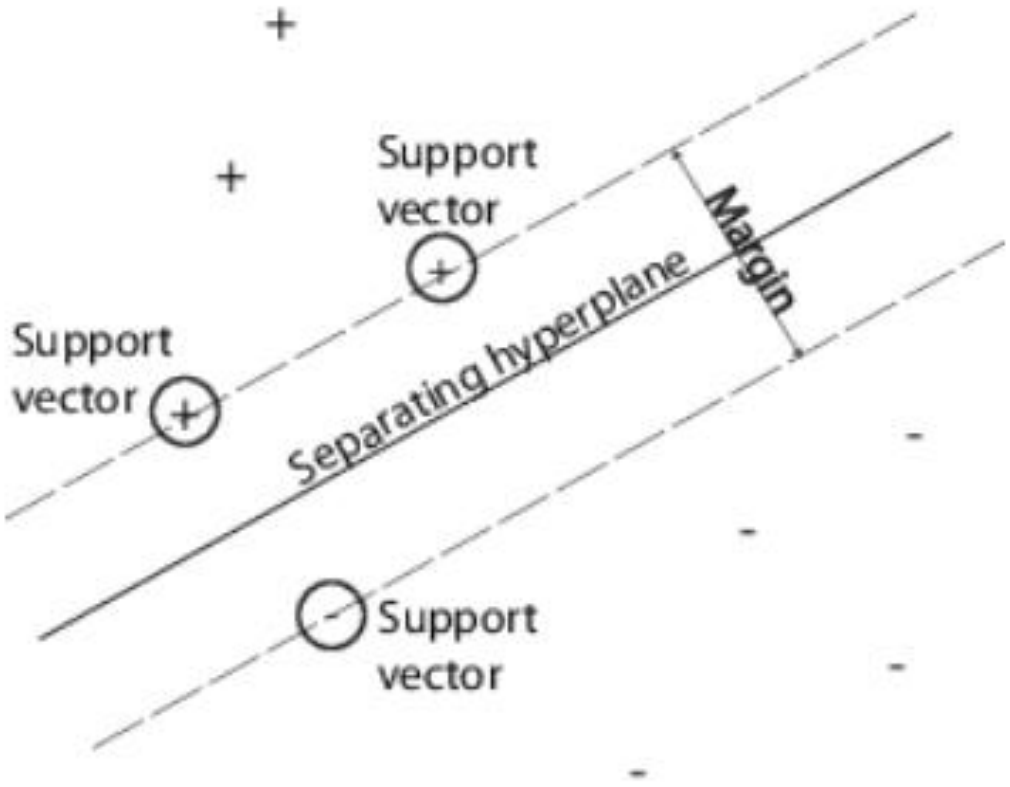
- Margin là $$M = \frac{2}{\|\beta\|}$$

- Cực đại M bằng cách tìm $\|\boldsymbol{\beta}\|$ cực tiểu với ràng buộc

$$y_i(\beta^T x_i + \beta_0) \geq 1 \ \forall i$$

$\mathbf{x}_i$ là data của lớp và y_i là nhãn, $y_i = \pm 1$

SUPPORT VECTOR MACHINE



SUPPORT VECTOR MACHINE

Cực tiểu hàm L dùng nhân tử Lagrange $\lambda_i >= 0$, ta giải bài toán đối ngẫu tìm λ_i cực đại $g(\boldsymbol{\lambda})$

$$L(\boldsymbol{\beta}, \beta_0, \boldsymbol{\lambda}) = 0.5\|\boldsymbol{\beta}\|^2 + \sum_{i=1}^{N} \lambda_i (1 - y_i(\boldsymbol{\beta}^T \boldsymbol{x} + \beta_0))$$

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = \boldsymbol{\beta} - \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i = 0 \rightarrow \boldsymbol{\beta} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i,$$

$$\frac{\partial L}{\partial \beta_0} = -\sum_{i=1}^{N} \lambda_i y_i = 0,$$

$$g(\boldsymbol{\lambda}) = \min L = \sum_{i=1}^{N} \lambda_i - 0.5 \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

$$\boldsymbol{V} = [y_1 \boldsymbol{x}_1 \;\; y_2 \boldsymbol{x}_2 \;\; y_3 \boldsymbol{x}_3 \ldots y_N \boldsymbol{x}_N], \boldsymbol{1} = [1 \; 1 \ldots 1]$$

$$g(\boldsymbol{\lambda}) = -0.5 \boldsymbol{\lambda}^T \boldsymbol{V}^T \boldsymbol{V} \boldsymbol{\lambda} + \boldsymbol{1} \boldsymbol{\lambda} = -0.5 \boldsymbol{\lambda}^T \boldsymbol{K} \boldsymbol{\lambda} + \boldsymbol{1} \boldsymbol{\lambda}$$

Hoặc ta tìm cực tiểu hàm $L_D(\boldsymbol{\lambda}) = 0.5 \boldsymbol{\lambda}^T \boldsymbol{K} \boldsymbol{\lambda} - \boldsymbol{1} \boldsymbol{\lambda}$

SUPPORT VECTOR MACHINE

- Cực tiểu hàm $L_D(\boldsymbol{\lambda})$=- $g(\boldsymbol{\lambda})$, max $g(\boldsymbol{\lambda})$=min $L(\boldsymbol{\beta},\beta_0)$
- $\boldsymbol{\lambda}$ là vector N chiều có giá trị ≥0, đa số λ_i=0, các giá trị λ_i >0 tương ứng với support vector
- Biết $\boldsymbol{\lambda}$ ta suy ra , với $\boldsymbol{x}_i$ là support vector, $\boldsymbol{N}$ là số support vector, thường là ba

$$\boldsymbol{\beta} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i$$

và $\beta_0 = y_i - \boldsymbol{\beta}^T \boldsymbol{x}_i$

- Đây là bài toán qui hoạch lồi có thể giải bằng hàm quadprog của Matlab (hay các chương trình khác)

$\boldsymbol{\lambda}$ = quadprog($\boldsymbol{K}$, $\boldsymbol{f}$, $\boldsymbol{A}$, $\boldsymbol{a}$, $\boldsymbol{B}$, $\boldsymbol{b}$) giải bài toán cực tiểu

min 0.5*$\boldsymbol{\lambda}$'*$\boldsymbol{K}$* $\boldsymbol{\lambda}$ + $\boldsymbol{f}$* λ với ràng buộc $\boldsymbol{A}$* $\boldsymbol{\lambda}$ <= $\boldsymbol{a}$, $\boldsymbol{B}$* $\boldsymbol{\lambda}$ = $\boldsymbol{b}$.

SUPPORT VECTOR MACHINE MATLAB

```
close all;clc;
rng default
%generate 20 data points 2 D normal distribution
means = [2  2; 4  2];
cov = [.3 .2; .2 .3];
N1 = 10;N=2*N1;
X0 = mvnrnd(means(1,:),cov,N1);
X1 = mvnrnd(means(2,:),cov,N1);
X=cat(1,X0,X1);
y=cat(1,ones(N1,1),-ones(N1,1));
axis([0 6 0 6]), hold on;
for i=1:size(y)
    if y(i)==1
```

SUPPORT VECTOR MACHINE MATLAB

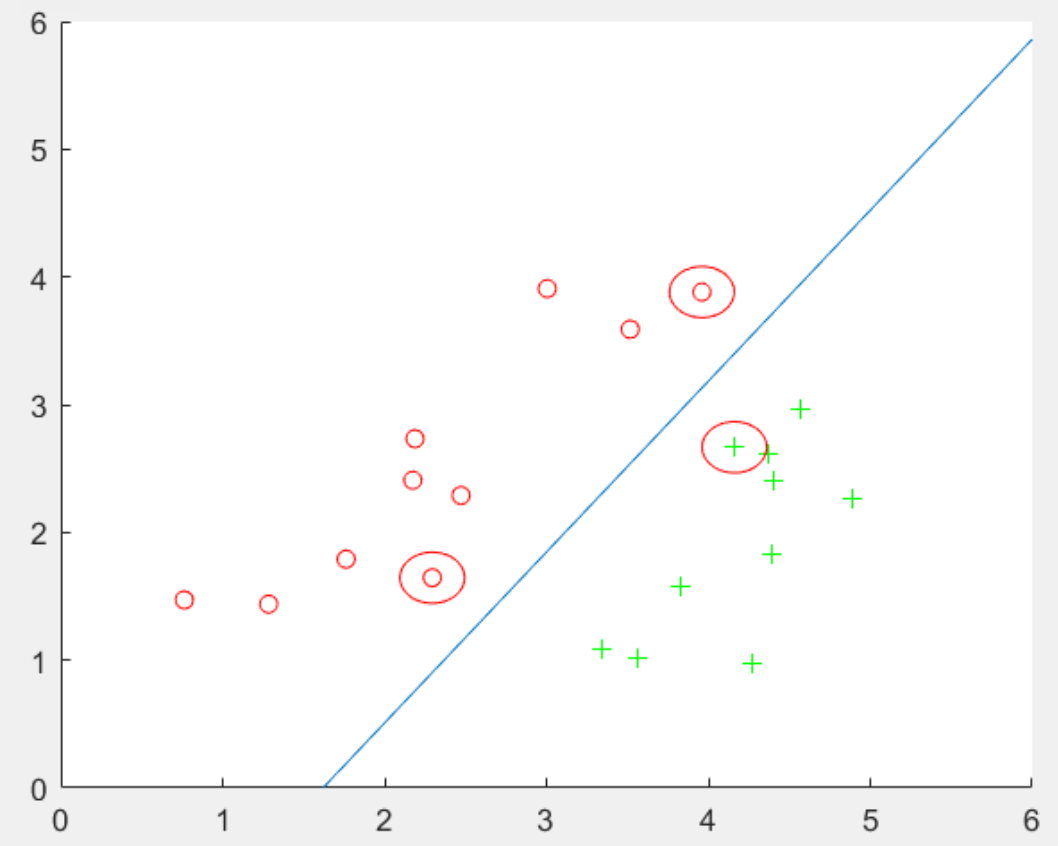
```
 plot(X(i,1),X(i,2),‘ro')
    else
        plot(X(i,1),X(i,2),‘g+')
    end
end
V=X.*y;
K=V*V';
f=-ones(N,1);
A=-eye(N);
a=zeros(N,1);
```

SUPPORT VECTOR MACHINE MATLAB

```
B=[y';zeros(N-1,N)];
b=zeros(N,1);
lambda=round(quadprog(K+eye(N)*0.001,f,A,a,B,b),3)
beta=(lambda.*y)'*X
for i=1:size(lambda)
    if (lambda(i)> 0)
        circle(X(i,:),0.2,'r')
    end
end
for i=1:size(lambda)
```

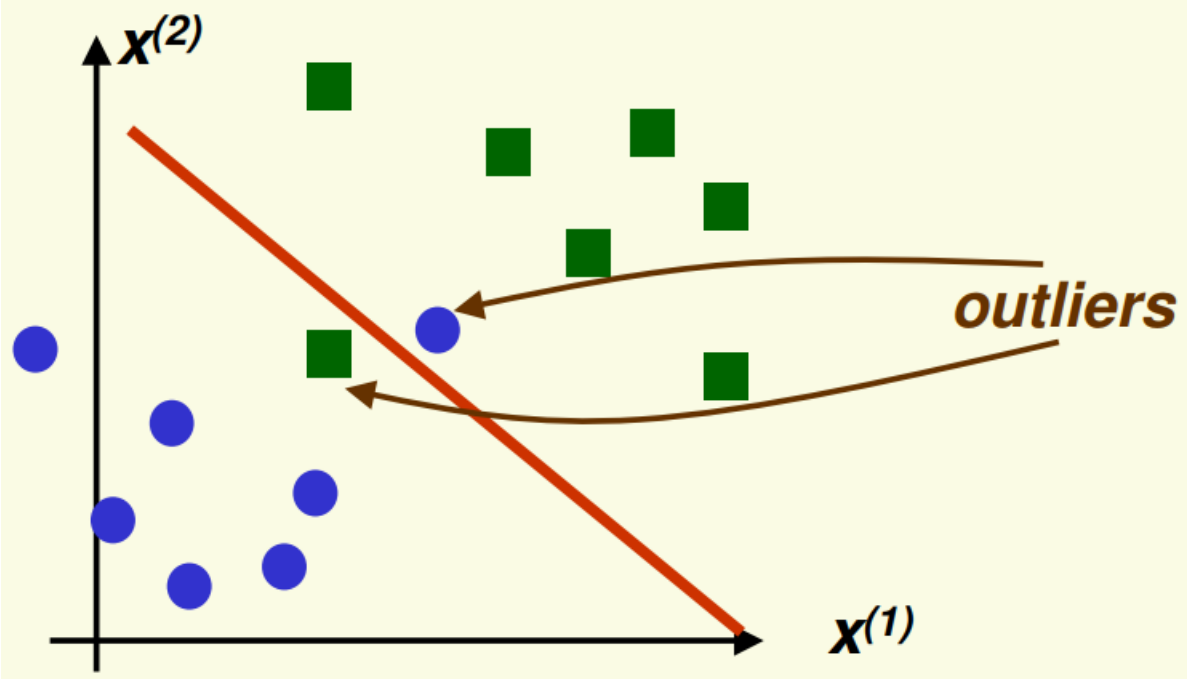
SUPPORT VECTOR MACHINE MATLAB

```
 if ~(lambda(i)== 0)
        beta0=y(i)-
beta*[X(i,:)]'
        break
    end
end
u=1:6;
v=(-beta0-
beta(1)*u)/beta(2);
plot(u,v)
```



SVM NOT LINEAR SEPARABLE

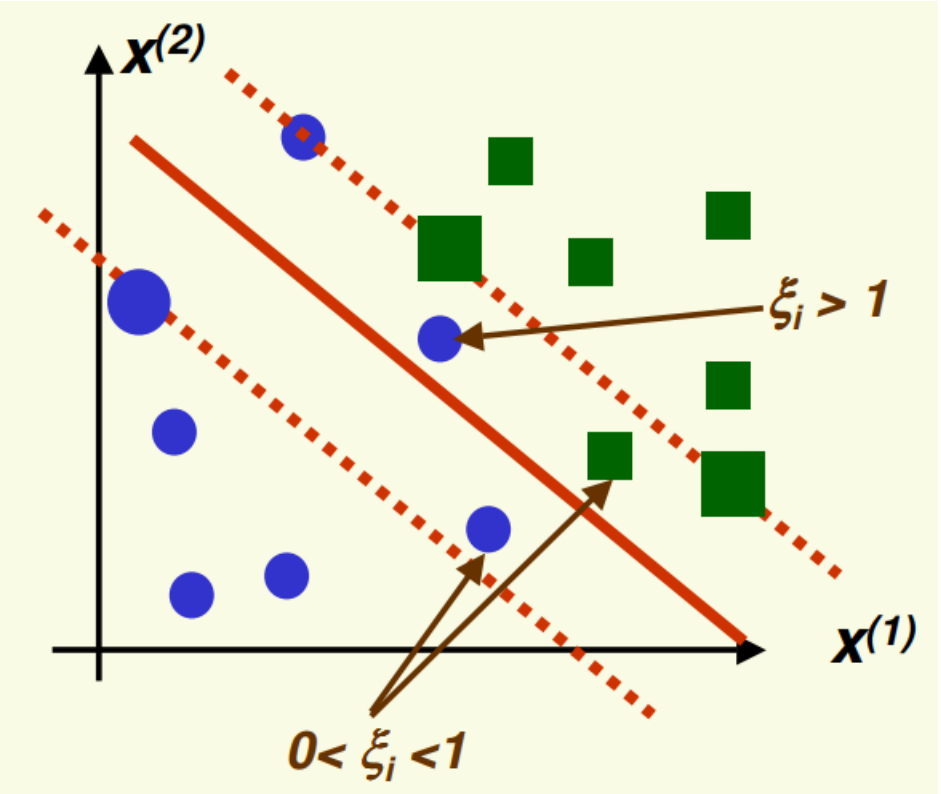
Trường hợp này các dữ liệu xâm lấn nhau, ta có thể phân lớp bằng hàm tuyến tính khi hy sinh vài dữ liệu bị phân lớp sai, gọi là soft margin.



SVM NOT LINEAR SEPARABLE

- Với mỗi điểm dữ liệu x_i ta gán biến slack ξ_i
- Ràng buộc $y_i(\beta^T x_i + \beta_0) \geq 1 \ \forall i$ đổi thành $y_i(\boldsymbol{\beta}^T \boldsymbol{x}_i + \beta_0) \geq 1 - \xi_i$

- Nếu $\xi_i > 1$ là phân lớp sai, $\xi_i = 0$: phân lớp đúng, $0 < \xi_i < 1$ phân lớp đúng nhưng điểm dữ liệu nằm trong vùng margin, tức là gần đường phân lớp hơn support vector.

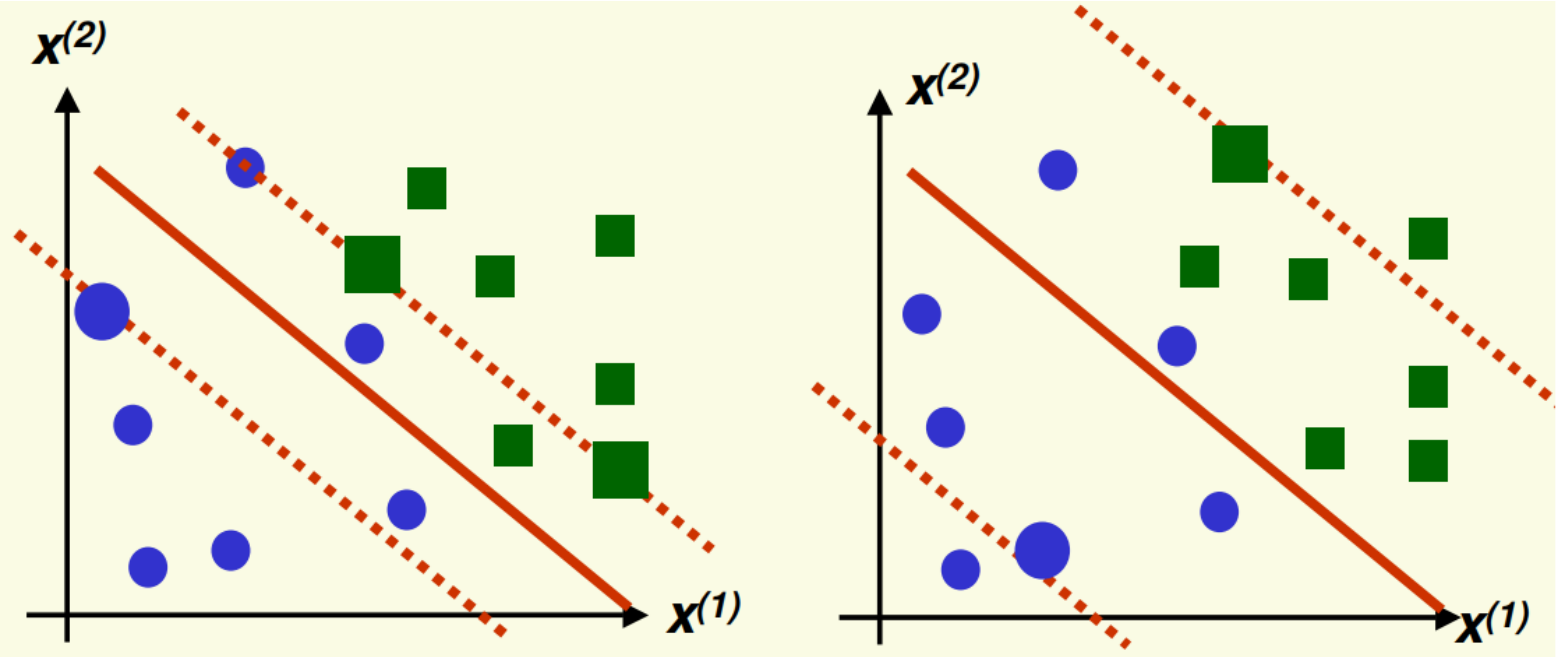


SVM NOT LINEAR SEPARABLE

- Cực tiểu hàm $L(\boldsymbol{\beta}, \boldsymbol{\xi})=0.5\| \boldsymbol{\beta} \|^2+C\sum \xi_i$, C là hằng số dương

Với ràng buộc $\xi_i \geq 1- y_i(\boldsymbol{\beta}^T \boldsymbol{x}_i +\beta_0)$, $\xi_i \geq 0$,

- Nếu chọn C lớn ξ_i sẽ nhỏ , có ít điểm bị phân loại sai hay rơi vào vùng margin. Hình dưới bên trái tương ứng C lớn còn hình bên phải là C nhỏ.



- Bài toán đối ngẫu là cực đại

$$L_D(\boldsymbol{\lambda}) = 0.5\boldsymbol{\lambda}^T \boldsymbol{K}\boldsymbol{\lambda} - \boldsymbol{1}\boldsymbol{\lambda}$$

Với ràng buộc

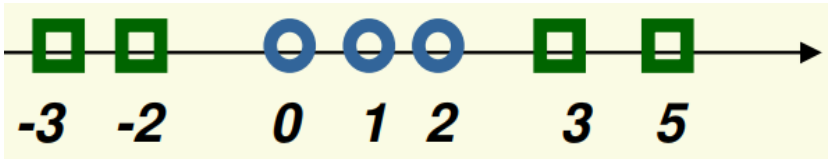
$$\sum_{i=1}^{N} \lambda_i y_i = 0, 0 \le \lambda_i \le C$$

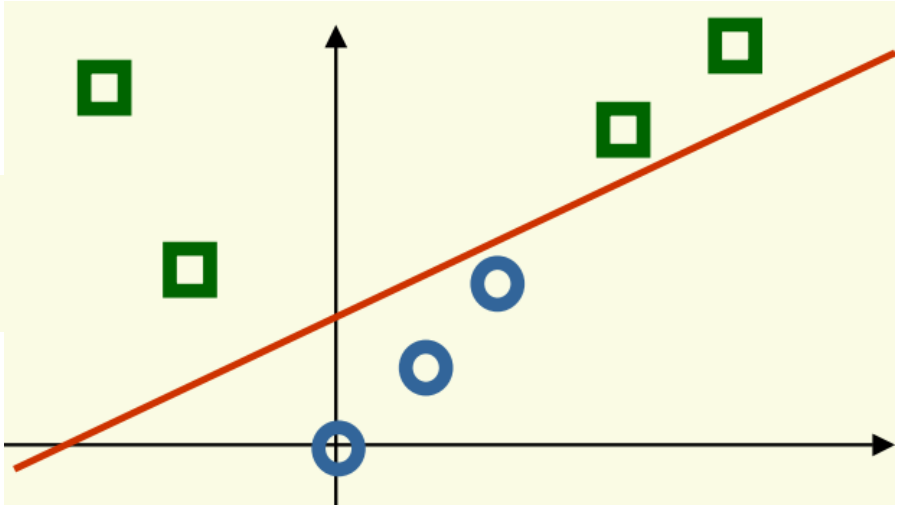
- Như vậy bài toán phân lớp non linearly separable đưa về bài toán linearly separable với một số điểm dữ liệu có thể phân lớp sai

SVM NONLINEAR

Đối với bài toán không phân chia tuyến tính được not linearly separable ta dùng phương pháp kernel biến đổi dữ liệu lên không gian có số chiều lớn hơn phân chia tuyến tính theo định lý Cover.

Ví dụ xét dữ liệu một chiều phân chia tuyến tính, ta biến đổi thành dữ liệu 2 chiều $\varphi(x)=(x,x^2)$, tìm đường thẳng phân chia, sau đó biến đổi đường thẳng phân chia trở lại không gian cũ





SVM NONLINEAR

Trở lại hàm Lagrange

$$g(\boldsymbol{\lambda}) = \min L = \sum_{i=1}^{N} \lambda_i - 0.5 \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

Dùng phép biến đổi $\mathbf{x} \rightarrow \varphi(\mathbf{x})$

$$g(\boldsymbol{\lambda}) = \min L = \sum_{i=1}^{N} \lambda_i - 0.5 \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \varphi^T(\boldsymbol{x}_i) \varphi(\boldsymbol{x}_j)$$

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \varphi^T(\boldsymbol{x}_i) \varphi(\boldsymbol{x}_j)$$

$$g(\boldsymbol{\lambda}) = \min L = \sum_{i=1}^{N} \lambda_i - 0.5 \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

K gọi là kernel, có tính chất đối xứng và phải thỏa điều kiện Mercer

SVM NONLINEAR

$$\sum_{i=1}^{N}\sum_{j=1}^{N} K(x_i, x_j)c_i c_j \geq 0$$

Lúc đó

$$g(\lambda) = \min L = \sum_{i=1}^{N} \lambda_i - 0.5\lambda^T \, \mathrm{K} \, \lambda$$

K là ma trận N chiều đối xứng bán xác định dương có phần tử $k_{i,j} = y_i y_j K(\boldsymbol{x}_i, x_j)$

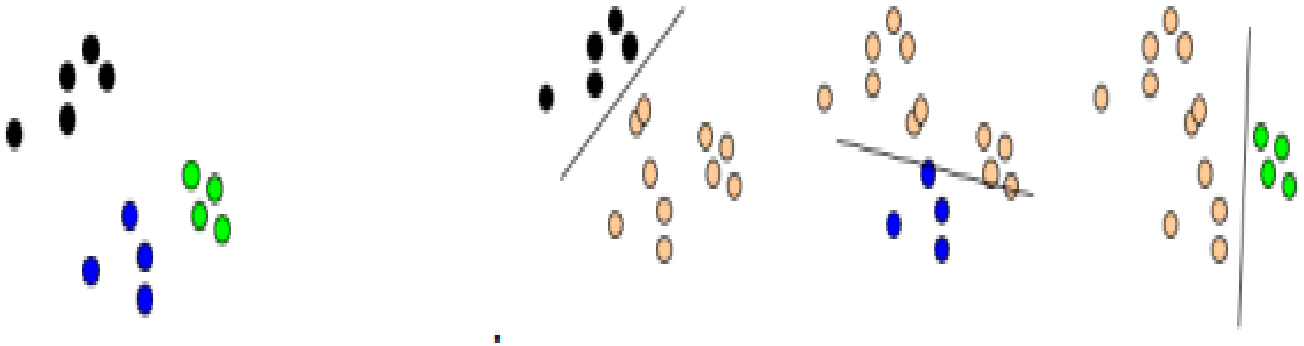
Các hàm kernel thông dụng là Linear, Polynomial, Radial Basic Function (RBF), Sigmoid

Tên	Công thức	kernel	Thiết lập hệ số
linear	$\mathbf{x}^T\mathbf{z}$	'linear'	không có hệ số
polynomial	$(r + \gamma\mathbf{x}^T\mathbf{z})^d$	'poly'	d: degree, γ: gamma, r: coef0
sigmoid	$\tanh(\gamma\mathbf{x}^T\mathbf{z} + r)$	'sigmoid'	γ: gamma, r: coef0
rbf	$\exp(-\gamma\|\|\mathbf{x} - \mathbf{z}\|\|_2^2)$	'rbf'	$\gamma > 0$: gamma

SVM MULTICLASS

- Nguyên tắc One versus All OVA, One vs Rest OvR

Gỉa sử cần phân dữ liệu ra k lớp, ta dùng k bộ phân lớp nhị phân. Huấn luyện lớp I với dữ liệu thuộc lớp I mang nhãn +1 còn các dữ liệu khác nhãn -1. Ta được k hàm phân lớp. Sau khi huấn luyện xong, dữ liệu mới đưa và sẽ được đánh giá bởi k bộ phân lớp. Bộ phân lớp j có điểm đánh giá cao nhất thì dữ liệu sẽ thuộc lớp j. Có thể có trường hợp điểm đánh giá bằng nhau và khó phân xử



SVM MULTICLASS

- **Nguyên tắc One vs One OvO**

Dùng k(k-1)/2 bộ phân lớp nhị phân, mỡi bộ phân lớp kij dùng để phân hai lớp I và j. Khi đưa dữ liệu vào ta xét đánh giá của các bộ phân lớp để phân xử. Có thể có trường hợp điểm đánh giá bằng nhau và khó phân xử

- Gỉa sử cần nhận dạng 10 số từ 0 đến 9 ta cần 10 bộ phân lớp OvA và 45 bộ phân lớp OvO.

- **Error-Correcting Output Codes**

Dùng nhiều bộ phân lớp nhị phân, giả sử phân lớp 10 số, dùng 15 bộ phân lớp f0..f14, ví dụ f0 phân tách nhóm (0; 2; 4; 6; 8) khỏi nhóm (1; 3; 5; 7; 9). Như vậy mỗi lớp có một mã 15 bit .không trùng nhau, tránh tình trạng mơ hờ khó xử.

SVM MULTICLASS

Class	Code Word														
	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
0	1	1	0	0	0	0	1	0	1	0	0	1	1	0	1
1	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0
2	1	0	0	1	0	0	0	1	1	1	1	0	1	0	1
3	0	0	1	1	0	1	1	1	0	0	0	0	1	0	1
4	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1
5	0	1	0	0	1	1	0	1	1	1	0	0	0	0	1
6	1	0	1	1	1	0	0	0	0	1	0	1	0	0	1
7	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1
8	1	1	0	1	0	1	1	0	0	1	0	0	0	1	1
9	0	1	1	1	0	0	0	0	1	0	1	0	0	1	1

SVM MULTICLASS

SVM OPENCV 3.4 C++

```
#include "stdafx.h"
#include <opencv2/opencv.hpp>
using namespace cv;
using namespace cv::ml;
int main(int, char**)
{// Set up training data
int labels[4] = { 1, -1, -1, -1 };
float trainingData[4][2] = { { 501, 10 },{ 255, 10 },{ 501, 255 },{ 10, 501 } };
Mat trainingDataMat(4, 2, CV_32F, trainingData);
Mat labelsMat(4, 1, CV_32SC1, labels);
// Train the SVM
Ptr<SVM> svm = SVM::create();
```

SVM OPENCV 3.4 C++

```
svm->setType(SVM::C_SVC);
svm->setKernel(SVM::LINEAR);
svm->setTermCriteria (TermCriteria(TermCriteria::MAX_ITER, 100, 1e-6));
svm->train(trainingDataMat, ROW_SAMPLE, labelsMat);
// Data for visual representation
int width = 512, height = 512;
Mat image = Mat::zeros(height, width, CV_8UC3);
// Show the decision regions given by the SVM
Vec3b green(0, 255, 0), blue(255, 0, 0);
for (int i = 0; i < image.rows; i++)
```

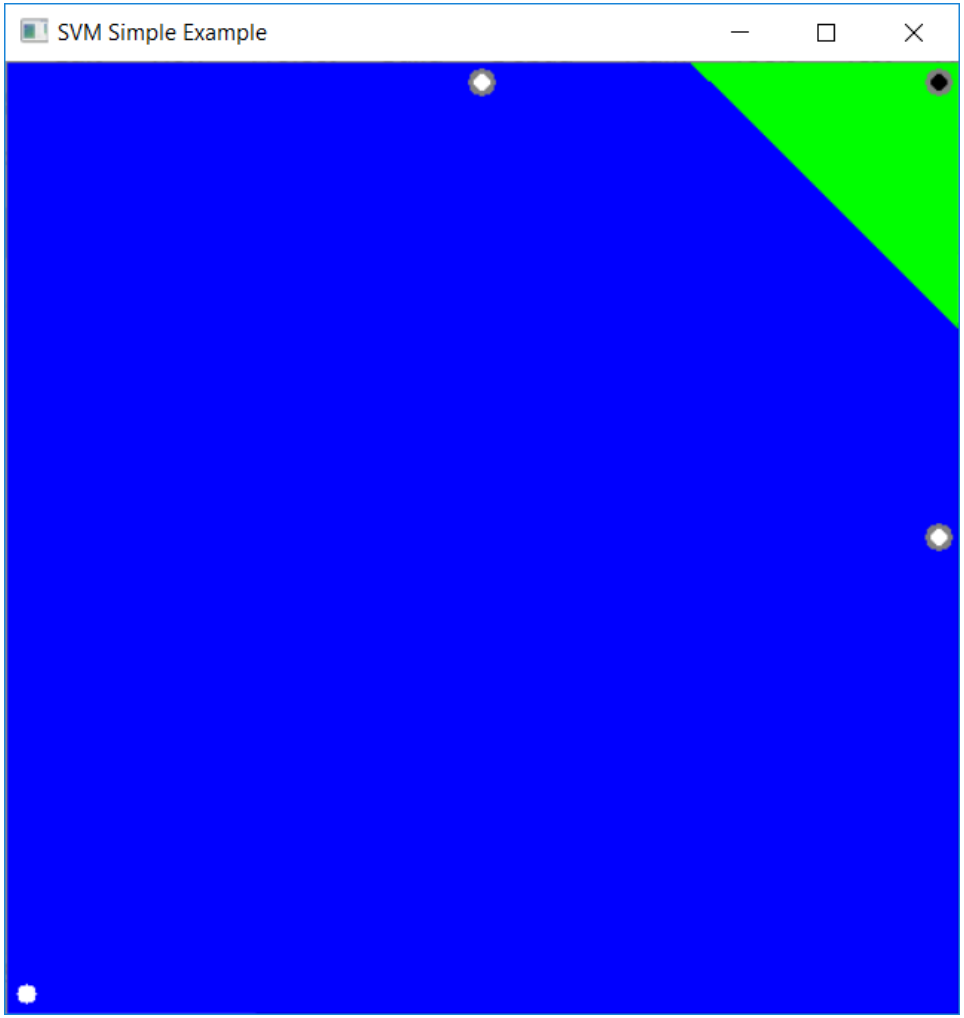
SVM OPENCV 3.4 C++

```
{ for (int j = 0; j < image.cols; j++)
{ Mat sampleMat = (Mat_<float>(1, 2) << j, i);
float response = svm->predict(sampleMat);
if (response == 1)
image.at<Vec3b>(i, j) = green;
else if (response == -1)
image.at<Vec3b>(i, j) = blue; }}
// Show the training data
int thickness = -1;
circle(image, Point(501, 10), 5, Scalar(0, 0, 0), thickness);
circle(image, Point(255, 10), 5, Scalar(255, 255, 255),
thickness);
```

SVM OPENCV 3.4 C++

```
circle(image, Point(501, 255), 5, Scalar(255, 255, 255), thickness);
circle(image, Point(10, 501), 5, Scalar(255, 255, 255), thickness);
// Show support vectors
thickness = 2;
Mat sv = svm->getUncompressedSupportVectors();
for (int i = 0; i < sv.rows; i++)
{ const float* v = sv.ptr<float>(i);
circle(image, Point((int)v[0], (int)v[1]), 6, Scalar(128, 128, 128), thickness);}
imwrite("result.png", image);        // save the image
imshow("SVM Simple Example", image); // show it to the user
waitKey(); return 0; }
```

SVM OPENCV 3.4 C++



SVM OPENCV 3.4 C++

Set up the training data:

The training data of this example is formed by a set of labeled 2D-points that belong to one of two different classes; one of the classes consists of one point and the other of three points.

```
int labels[4] = {1, -1, -1, -1};
float trainingData[4][2] = { {501, 10}, {255, 10}, {501, 255}, {10, 501} };
```

The function cv::ml::SVM::train that will be used afterwards requires the training data to be stored as cv::Mat objects of floats. Therefore, we create these objects from the arrays defined above:

```
Mat trainingDataMat(4, 2, CV_32F, trainingData);
Mat labelsMat(4, 1, CV_32SC1, labels);
```

SVM OPENCV 3.4 C++

Set up SVM's parameters

SVMs can be used in a wide variety of problems (e.g. problems with non-linearly separable data, a SVM using a kernel function to raise the dimensionality of the examples, etc). As a consequence of this, we have to define some parameters before training the SVM. These parameters are stored in an object of the class cv::ml::SVM.

```
Ptr<SVM> svm = SVM::create();
    svm->setType(SVM::C_SVC);
    svm->setKernel(SVM::LINEAR);
    svm->setTermCriteria
(TermCriteria(TermCriteria::MAX_ITER, 100, 1e-6));
```

SVM OPENCV 3.4 C++

Type of SVM. We choose here the type C_SVC that can be used for n-class classification (n ≥ 2).

Type of SVM kernel. It is a mapping done to the training data to improve its resemblance to a linearly separable set of data. This mapping consists of increasing the dimensionality of the data and is done efficiently using a kernel function. We choose here the type LINEAR which means that no mapping is done. This parameter is defined using cv::ml::SVM::setKernel.

Termination criteria of the algorithm. The SVM training procedure is implemented solving a constrained quadratic optimization problem in an iterative fashion. This parameter is defined in a structure cv::TermCriteria .

Train the SVM We call the method cv::ml::SVM::train to build the SVM model.

svm->train(trainingDataMat, ROW_SAMPLE, labelsMat);

SVM OPENCV 3.4 C++

Regions classified by the SVM

The method cv::ml::SVM::predict is used to classify an input sample using a trained SVM. An image is interpreting its pixels as points of the Cartesian plane. Each of the points is colored depending on the class predicted by the SVM; in green if it is the class with label 1 and in blue if it is the class with label -1.

```
Vec3b green(0,255,0), blue(255,0,0);
for (int i = 0; i < image.rows; i++)
   { for (int j = 0; j < image.cols; j++)
    {Mat sampleMat = (Mat_<float>(1,2) << j,i);
     float response = svm->predict(sampleMat);
      if (response == 1) image.at<Vec3b>(i,j)  = green;
     else if (response == -1)  image.at<Vec3b>(i,j)  = blue;}}
```

SVM OPENCV 3.4 C++

Support vectors

The method cv::ml::SVM::getSupportVectors obtain all of the support vectors. We have used this methods here to find the training examples that are support vectors and highlight them.

```
thickness = 2;
Mat sv = svm->getUncompressedSupportVectors();
for (int i = 0; i < sv.rows; i++)
{
    const float* v = sv.ptr<float>(i);
    circle(image,  Point( (int) v[0], (int) v[1]), 6, Scalar(128, 128, 128), thickness);
}
```

SVM PYTHON

```
import cv2 as cv

import numpy as np

# Set up training data

labels = np.array([1, -1, -1, -1])

trainingData = np.matrix([[501, 10], [255, 10], [501, 255], [10, 501]], dtype=np.float32)

# Train the SVM

svm = cv.ml.SVM_create()

svm.setType(cv.ml.SVM_C_SVC)

svm.setKernel(cv.ml.SVM_LINEAR)

svm.setTermCriteria((cv.TERM_CRITERIA_MAX_ITER, 100, 1e-6))

svm.train(trainingData, cv.ml.ROW_SAMPLE, labels)
```

SVM PYTHON

```
# Data for visual representation
width = 512
height = 512
image = np.zeros((height, width, 3), dtype=np.uint8)
# Show the decision regions given by the SVM
green = (0,255,0)
blue = (255,0,0)
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        sampleMat = np.matrix([[j,i]], dtype=np.float32)
         response = svm.predict(sampleMat)[1]
        if response == 1:
            image[i,j] = green
```

SVM PYTHON

```
        elif response == -1:
        image[i,j] = blue
# Show the training data
thickness = -1
cv.circle(image, (501,  10), 5, (  0,   0,   0), thickness)
cv.circle(image, (255,  10), 5, (255, 255, 255), thickness)
cv.circle(image, (501, 255), 5, (255, 255, 255), thickness)
cv.circle(image, ( 10, 501), 5, (255, 255, 255), thickness)
# Show support vectors
thickness = 2
```

SVM PYTHON

```
sv = svm.getUncompressedSupportVectors()
for i in range(sv.shape[0]):
    cv.circle(image, (sv[i,0], sv[i,1]), 6, (128, 128, 128), thickness)
cv.imwrite('result.png', image) # save the image
cv.imshow('SVM Simple Example', image) # show it to the user
cv.waitKey()
```

SVM MATLAB

1/Train classifier

SVMModel = fitcsvm(X,Y,'KernelFunction','rbf',...

‘Standardize',true,'ClassNames',{'negClass','posClass'});

X — Matrix of predictor data, where each row is one observation, and each column is one predictor.

Y — Array of class labels with each row corresponding to the value of the corresponding row in X. Y can be a categorical, character, or string array, a logical or numeric vector, or a cell array of character vectors.

KernelFunction — The default value is 'linear' for two-class learning, which separates the data by a hyperplane. The value 'gaussian' (or 'rbf') is the default for one-class learning, and specifies to use the Gaussian (or radial basis function) kernel.

Standardize — Flag indicating whether the software should standardize the predictors before training the classifier.

ClassNames — Distinguishes between the negative and positive classes, or specifies which classes to include in the data. The negative class is the first element (or row of a character array), e.g., 'negClass', and the positive class is the second element (or row of a character array), e.g., 'posClass'. ClassNames must be the same data type as Y. It is good practice to specify the class names, especially if you are comparing the performance of different classifiers.

2/Classifying New Data with an SVM Classifier

Classify new data using predict. The syntax for classifying new data using a trained SVM classifier (SVMModel) is:

[label,score] = predict(SVMModel,newX);

The resulting vector, label, represents the classification of each row in X. score is an n-by-2 matrix of soft scores. Each row corresponds to a row in X, which is a new observation. The first column contains the scores for the observations being classified in the negative class, and the second column contains the scores observations being classified in the positive class.

Generate 100 points uniformly distributed in the unit disk. To do so, generate a radius r as the square root of a uniform random variable, generate an angle t uniformly in (0,), and put the point at (r cos(t), r sin(t)). Generate 100 points uniformly distributed in the annulus. The radius is again proportional to a square root, this time a square root of the uniform distribution from 1 through 4.

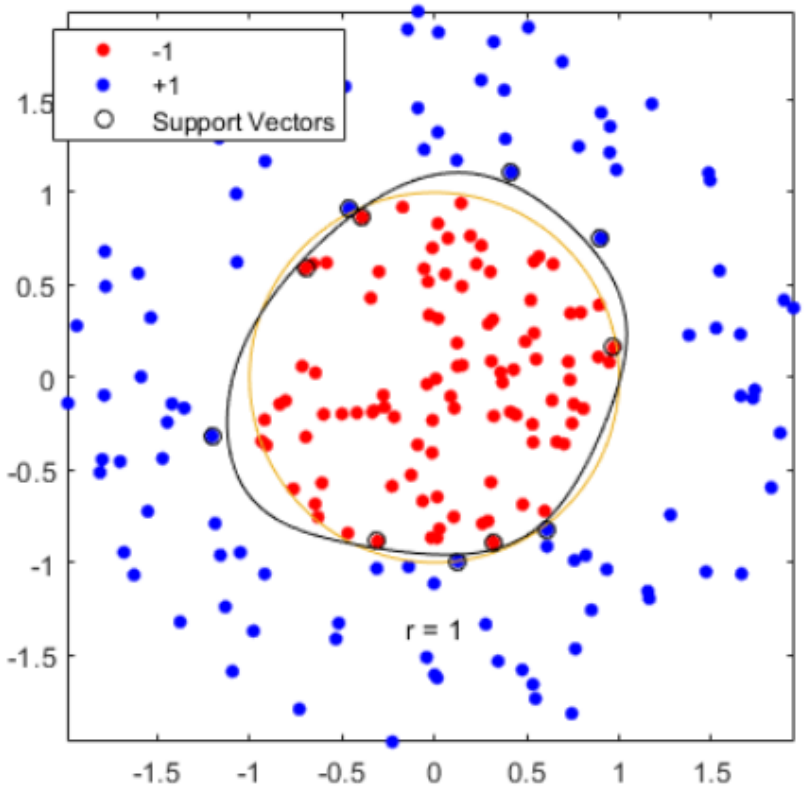
```
rng(1); % For reproducibility
r = sqrt(rand(100,1)); % Radius
t = 2*pi*rand(100,1);  % Angle
data1 = [r.*cos(t), r.*sin(t)]; % Points
r2 = sqrt(3*rand(100,1)+1); % Radius
t2 = 2*pi*rand(100,1);      % Angle
data2 = [r2.*cos(t2), r2.*sin(t2)]; % points
```

```
figure;
plot(data1(:,1),data1(:,2),'r.','MarkerSize',15)
hold on
plot(data2(:,1),data2(:,2),'b.','MarkerSize',15)
ezpolar(@(x)1);ezpolar(@(x)2);
axis equal
hold off
data3 = [data1;data2]; theclass = ones(200,1); theclass(1:100) = -1;
```

Train an SVM classifier with KernelFunction set to 'rbf' and BoxConstraint set to Inf. Plot the decision boundary and flag the support vectors.

```
%Train the SVM Classifier
cl = fitcsvm(data3,theclass,'KernelFunction','rbf',...
    'BoxConstraint',Inf,'ClassNames',[-1,1]);
% Predict scores over the grid
d = 0.02;
[x1Grid,x2Grid] = meshgrid(min(data3(:,1)):d:max(data3(:,1)),...
    min(data3(:,2)):d:max(data3(:,2)));
xGrid = [x1Grid(:),x2Grid(:)];
[~,scores] = predict(cl,xGrid);
% Plot the data and the decision boundary
figure;
h(1:2) = gscatter(data3(:,1),data3(:,2),theclass,'rb','.');
```

```
hold on
ezpolar(@(x)1);
h(3) =
plot(data3(cl.IsSupportVector,1
),data3(cl.IsSupportVector,2),'k
o');
contour(x1Grid,x2Grid,reshape
(scores(:,2),size(x1Grid)),[0
0],'k');
legend(h,{'-1','+1','Support
Vectors'});
axis equal
hold off
```



Train SVM Classifier Using Custom Kernel

This example shows how to use a custom kernel function, such as the sigmoid kernel, to train SVM classifiers, and adjust custom kernel function parameters.

Generate a random set of points within the unit circle. Label points in the first and third quadrants as belonging to the positive class, and those in the second and fourth quadrants in the negative class.

```
rng(1);  % For reproducibility
n = 100; % Number of points per quadrant
r1 = sqrt(rand(2*n,1));                  % Random radii
t1 = [pi/2*rand(n,1); (pi/2*rand(n,1)+pi)]; % Random angles for Q1 and Q3
X1 = [r1.*cos(t1) r1.*sin(t1)];  % Polar-to-Cartesian conversion
```

```
r2 = sqrt(rand(2*n,1));
t2 = [pi/2*rand(n,1)+pi/2; (pi/2*rand(n,1)-pi/2)]; % Random angles for Q2 and Q4
X2 = [r2.*cos(t2) r2.*sin(t2)];
X = [X1; X2];        % Predictors
Y = ones(4*n,1);
Y(2*n + 1:end) = -1; % Labels
figure;
gscatter(X(:,1),X(:,2),Y);
title('Scatter Diagram of Simulated Data')
```

Write a function that accepts two matrices in the feature space as inputs, and transforms them into a Gram matrix using the sigmoid kernel. Save this code as a file named mysigmoid on your MATLAB® path.

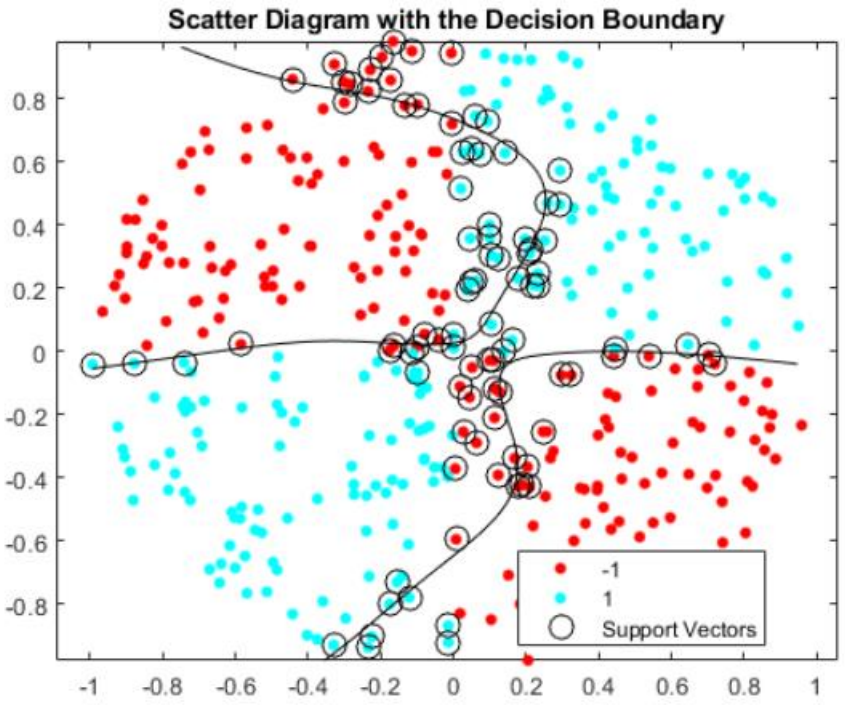
```
function G = mysigmoid(U,V)
% Sigmoid kernel function with slope gamma and intercept c
gamma = 1;
c = -1;
G = tanh(gamma*U*V' + c);
end
```

Train an SVM classifier using the sigmoid kernel function. It is good practice to standardize the data.

```
Mdl1 = itcsvm(X,Y,'KernelFunction','mysigmoid','Standardize',true);
```

```
 % Compute the scores over a grid
d = 0.02; % Step size of the grid
[x1Grid,x2Grid] = meshgrid(min(X(:,1)):d:max(X(:,1)),...
    min(X(:,2)):d:max(X(:,2)));
xGrid = [x1Grid(:),x2Grid(:)];        % The grid
[~,scores1] = predict(Mdl1,xGrid); % The scores
figure;
h(1:2) = gscatter(X(:,1),X(:,2),Y);
hold on
h(3) = plot(X(Mdl1.IsSupportVector,1),...
    X(Mdl1.IsSupportVector,2),'ko','MarkerSize',10);
```

```
% Support vectors
contour(x1Grid,x2Grid,reshape(scores1(:,2),size(x1Grid)),[0 0],'k');
 % Decision boundary
title('Scatter Diagram with the Decision Boundary')
legend({'-1','1','Support Vectors'},'Location','Best');
hold off
```



Determine the out-of-sample misclassification rate by using 10-fold cross validation.

```
CVMdl1 = crossval(Mdl1);

misclass1 = kfoldLoss(CVMdl1)
```

misclass1 = 0.1350

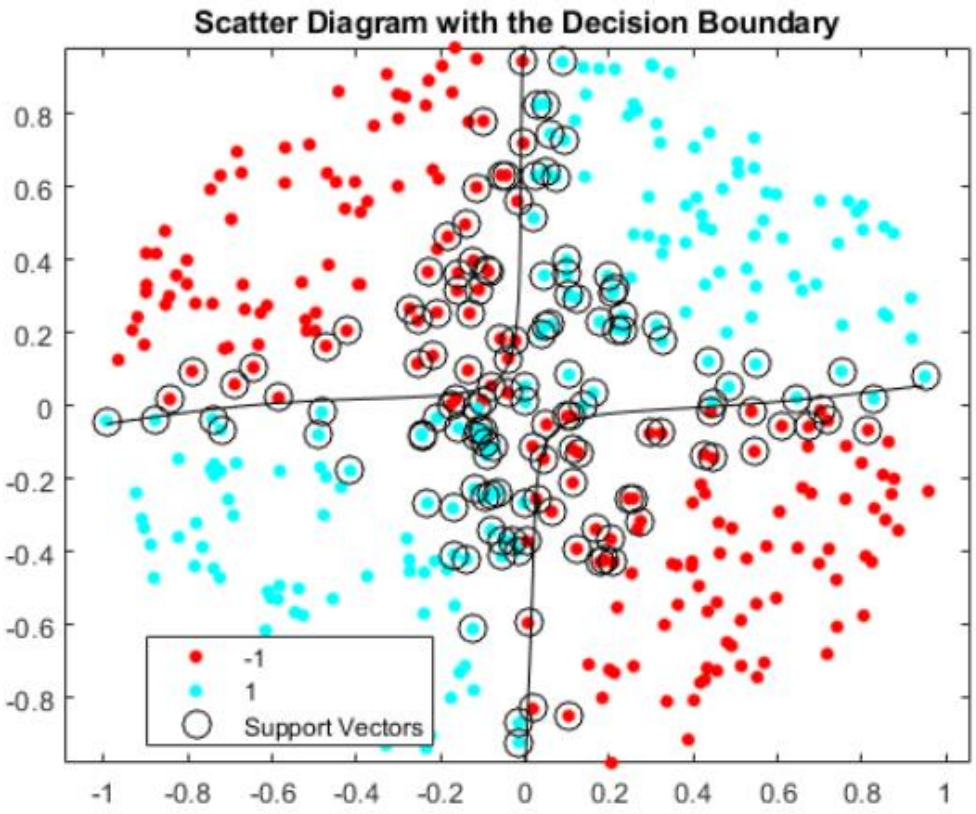
The out-of-sample misclassification rate is 13.5%.

Write another sigmoid function, but Set gamma = 0.5;.

```
function G = mysigmoid2(U,V)
% Sigmoid kernel function with slope gamma and intercept c
gamma = 0.5;
c = -1;
G = tanh(gamma*U*V' + c); end
```

Train another SVM classifier using the adjusted sigmoid kernel. Plot the data and the decision region, and determine the out-of-sample misclassification rate.

misclass2 = 0.0450



SVM MULTICLASS MATLAB

ClassificationECOC is an error-correcting output codes (ECOC) classifier for multiclass learning by reduction to multiple binary classifiers such as support vector machines (SVMs). Train a ClassificationECOC classifier using fitcecoc and the training data. Trained ClassificationECOC classifiers store the training data, parameter values, prior probabilities, and coding matrices. You can use these classifiers to: Estimate resubstitution predictions, Predict labels or posterior probabilities for new data.

Mdl = fitcecoc(Tbl,ResponseVarName) returns a full, trained, multiclass, error-correcting output codes (ECOC) model using the predictors in table Tbl and the class labels in Tbl.ResponseVarName. fitcecoc uses K(K – 1)/2 binary support vector machine (SVM) models using the one-versus-one coding design, where K is the number of unique class labels (levels). Mdl is a ClassificationECOC model.

Mdl = fitcecoc(Tbl,formula) returns an ECOC model using the predictors in table Tbl and the class labels. formula is an explanatory model of the response and a subset of predictor variables in Tbl used for training.

Mdl = fitcecoc(Tbl,Y) returns an ECOC model using the predictors in table Tbl and the class labels in vector Y.

Mdl = fitcecoc(X,Y) returns a trained ECOC model using the predictors X and the class labels Y.

Mdl = fitcecoc(___,Name,Value) returns an ECOC model with additional options specified by one or more Name,Value pair arguments, using any of the previous syntaxes.

For example, specify different binary learners, a different coding design, or to cross-validate. It is good practice to cross-validate using the Kfold Name,Value pair argument. The cross-validation results determine how well the model generalizes.

[Mdl,HyperparameterOptimizationResults] = fitcecoc(___,Name,Value) also returns hyperparameter optimization details when you pass an OptimizeHyperparameters name-value pair with Learners = 'linear'. For other Learners, the HyperparameterOptimizationResults property of Mdl contains the results. Hyperparameter optimization is not available for kernel binary learners.

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher. The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis.

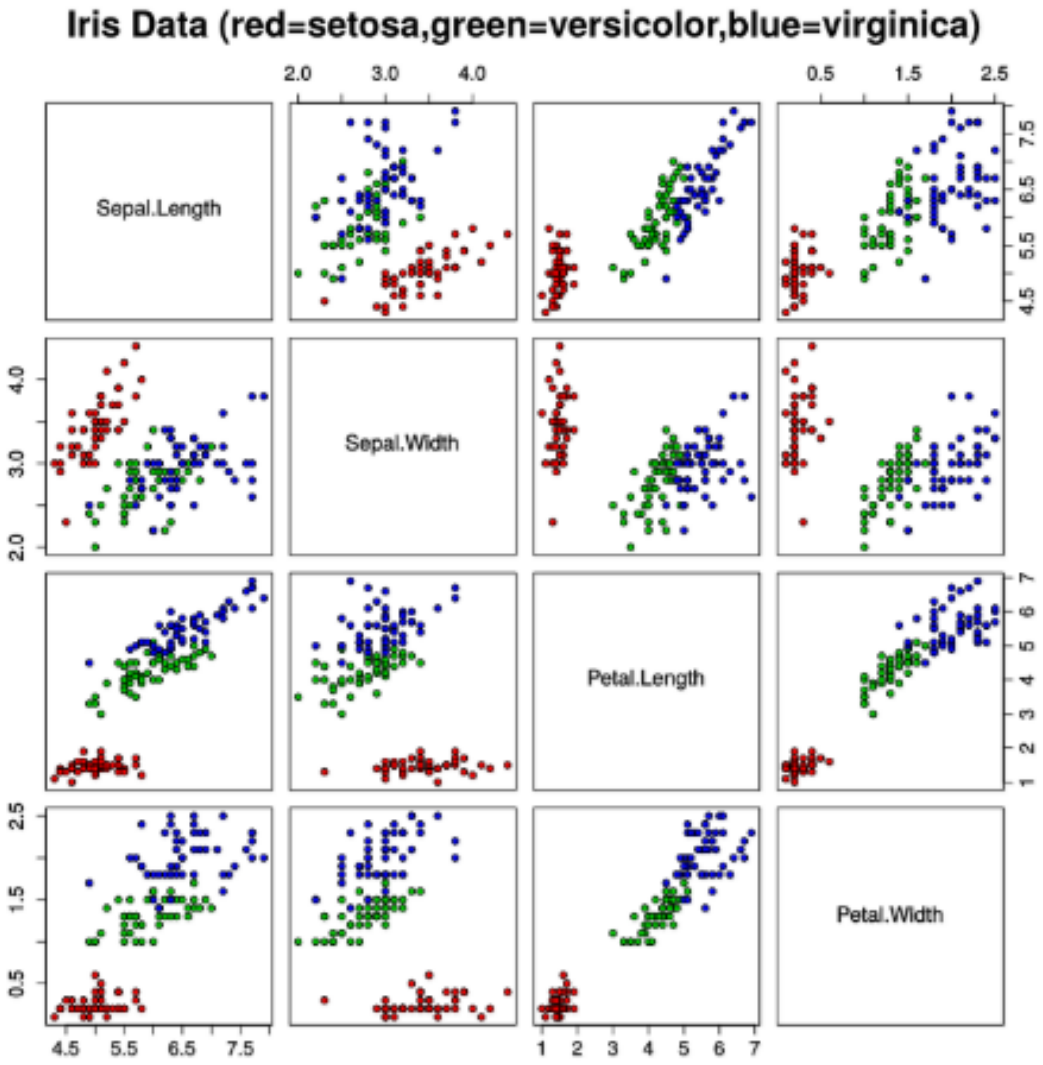
The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

Based on Fisher's linear discriminant model, this data set became a typical test case for many statistical classification techniques in machine learning such as support vector machines.

Iris setosa

Iris versicolor

Iris virginica


Iris Data (red=setosa,green=versicolor,blue=virginica)
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width

Train an error-correcting output codes (ECOC) multiclass model using support vector machine (SVM) binary learners with default options.

%Load Fisher's iris data set.

load fisheriris

X = meas;

Y = species;

Mdl = fitcecoc(X,Y)

Mdl is a ClassificationECOC model. By default, fitcecoc uses SVM binary learners, and uses a one-versus-one coding design. You can access Mdl properties using dot notation.

```
Mdl.ClassNames
ans = 3x1 cell array
    {'setosa'    }
    {'versicolor'}
    {'virginica' }
CodingMat = Mdl.CodingMatrix
CodingMat = 3×3
     1     1     0
    -1     0     1
     0    -1    -1
```

Compute the in-sample classification error.

isLoss = resubLoss(Mdl)

isLoss = 0.0067

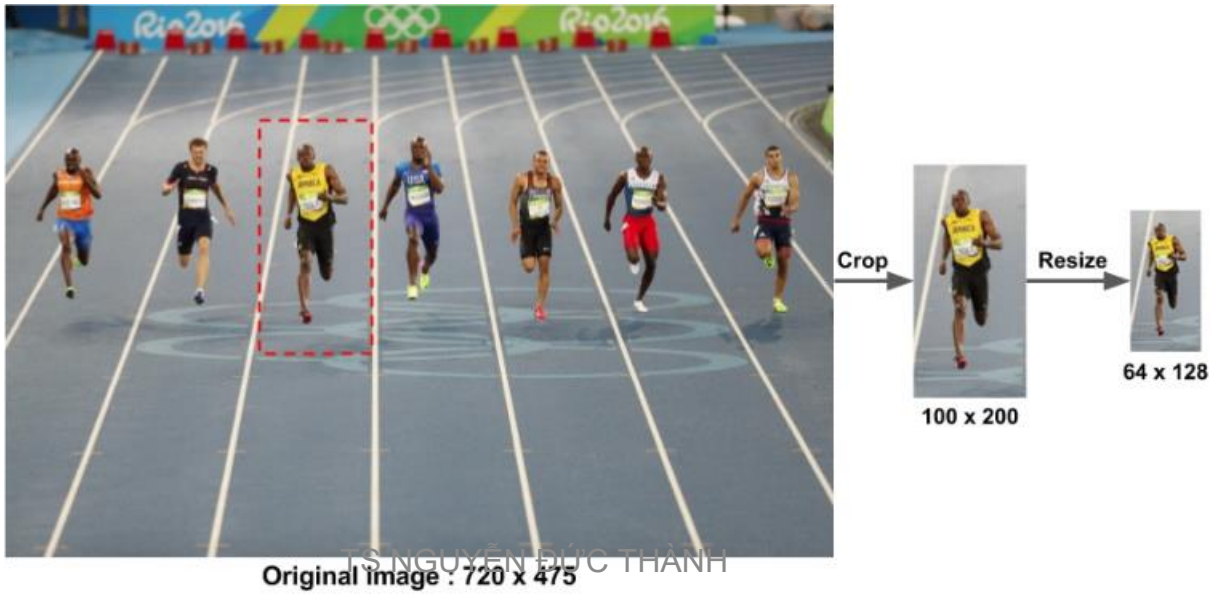
The classification error is small, but the classifier might have been overfit. You can cross-validate the classifier using crossval.

HISTOGRAM OF ORIENTED GRADIENT

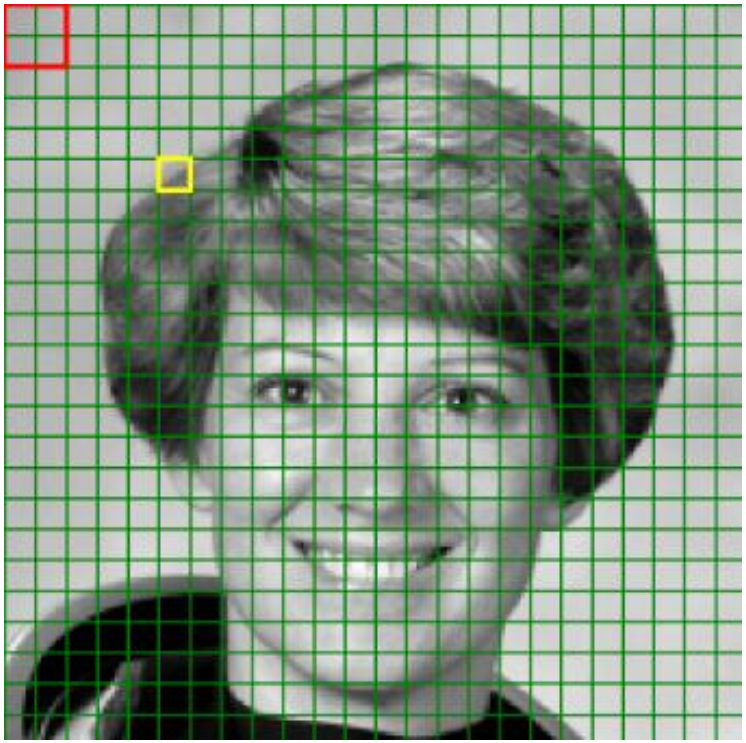
Navneet Dalal and Bill Triggs. *Histogram of oriented gradients for human detection*. 2005.

KHÁI NIỆM

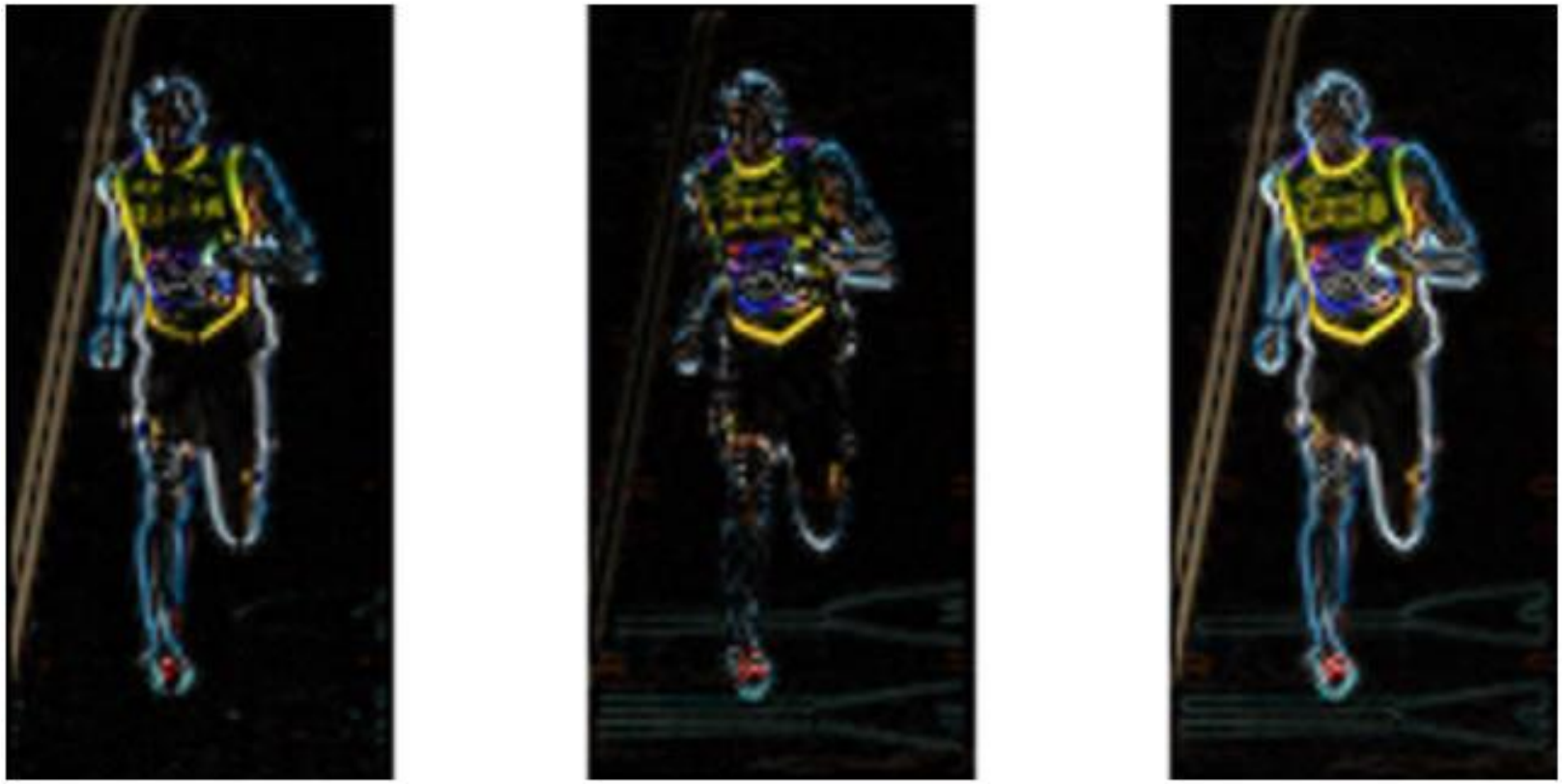
- Phương pháp HOG dùng gradient theo hướng để mô tả đặc trưng của đối tượng người đứng. Gradient có giá trị lớn ở biên và góc do đó mô tả đối tương chính xác. Dùng một cửa sổ trượt [-1 0 1] và $[-1\ 0\ 1]^T$ để tính gradient gx và gy ở mỗi pixel của vùng ảnh, sau đó tính số lần có cùng gradient [gx]+[gy] hay căn của gx^2+gy^2 theo các hướng $tg^{-1}([gy/gx])$ (histogram of oriented gradients). Ảnh có kích thước cao 128 và rộng 64.
- Hình ảnh được chia thành nhiều cell 8x8 và mỗi pixel được tính gradient theo độ lớn và hướng, sau đó histogram của cell được tính.



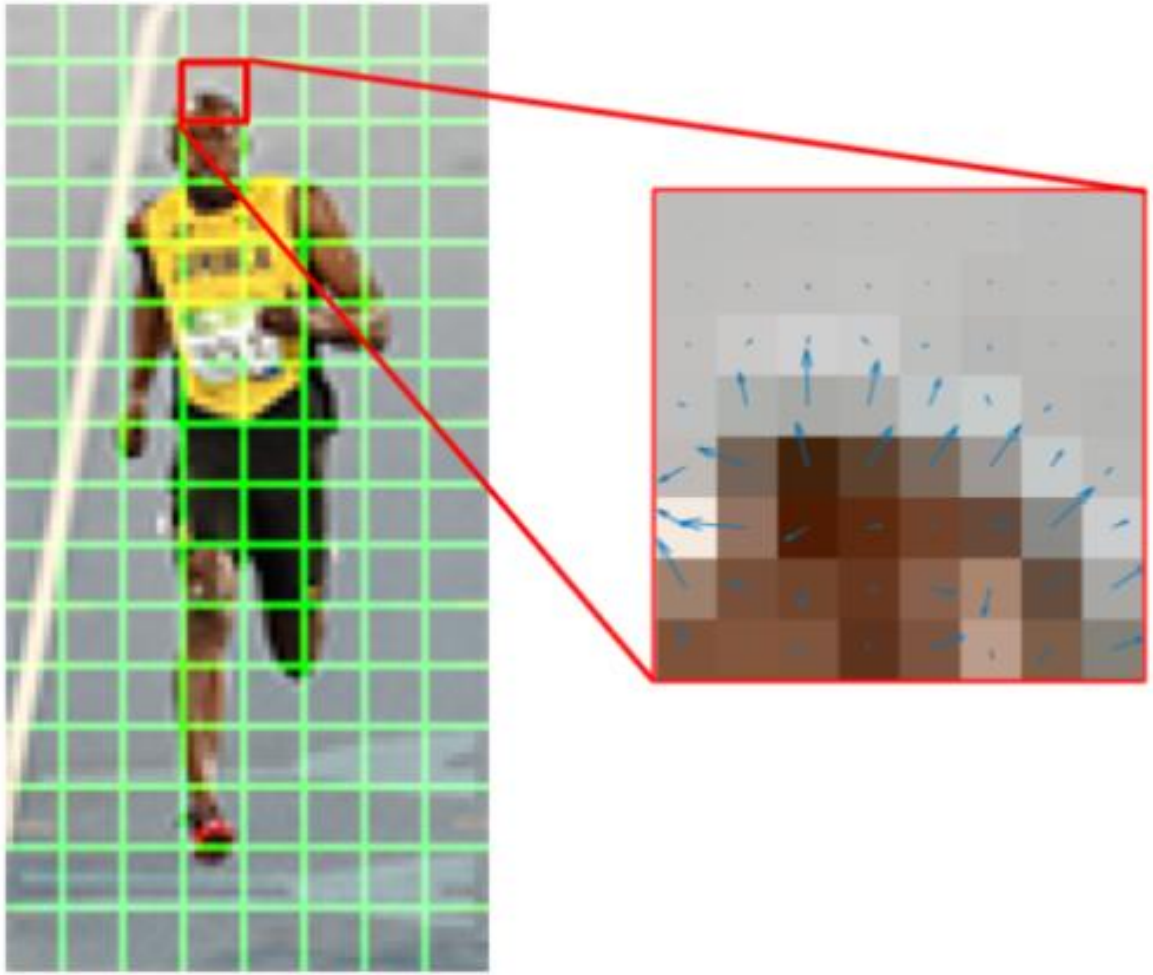
Tính gradient tại mỗi cell



0.01 74	0.01 109	0.02 45	0.04 40	0.07 42	0.04 130	0.08 35	0.14 48
0.00 76	0.02 79	0.04 44	0.06 45	0.03 58	0.04 63	0.12 59	0.15 62
0.01 70	0.04 63	0.07 52	0.02 22	0.02 74	0.09 53	0.13 51	0.12 82
0.04 57	0.06 55	0.03 81	0.01 105	0.08 51	0.12 60	0.07 59	0.01 157
0.05 56	0.04 36	0.02 138	0.06 58	0.12 44	0.09 45	0.03 106	0.03 137
0.05 51	0.01 29	0.03 63	0.12 62	0.12 50	0.04 20	0.02 98	0.03 56
0.01 20	0.01 102	0.08 55	0.13 59	0.06 88	0.01 104	0.04 46	0.02 131
0.01 124	0.04 92	0.12 61	0.09 45	0.04 148	0.02 48	0.05 11	0.06 67



Left : Absolute value of x-gradient. Center : Absolute value of y-gradient. Right : Magnitude of gradient.



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

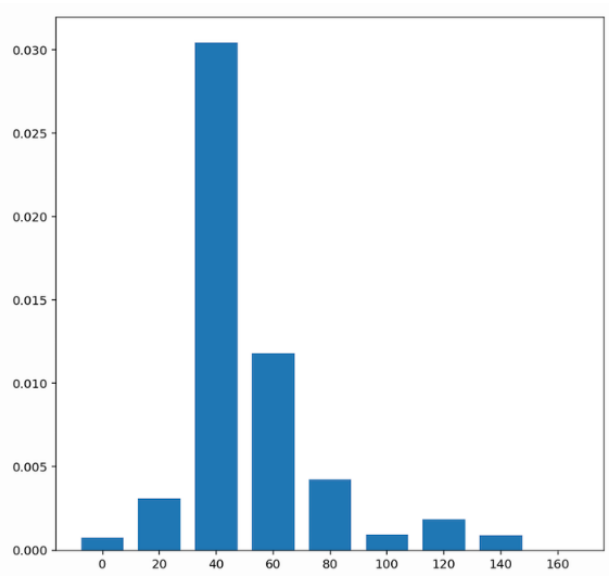
80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

KHÁI NIỆM

- Lượng tử hướng theo một số mức ví dụ 9, giá trị 0, 20^o , 40^o , ..,160^o mỗi phần tử của vec tơ là số pixel có cùng hướng (histogram) tuy nhiên mỗi pixel có trọng số khác nhau tùy theo suất của gradient ([gx]+[gy]. Cộng tất cả độ lớn của pixel có cùng hướng rồi chia cho 64 chứa vào số hạng tương ứng theo hướng của vector đặc trưng.

Thông tin này cho biết phân bố hướng của gradient của cell



0.01 60	0.01 100	0.02 40	0.04 40	0.07 40	0.04 120	0.08 20	0.14 40
0.00 60	0.02 60	0.04 40	0.06 40	0.03 40	0.04 60	0.12 40	0.15 60
0.01 60	0.04 60	0.07 40	0.02 20	0.02 60	0.09 40	0.13 40	0.12 80
0.04 40	0.06 40	0.03 80	0.01 100	0.08 40	0.12 60	0.07 40	0.01 140
0.05 40	0.04 20	0.02 120	0.06 40	0.12 40	0.09 40	0.03 100	0.03 120
0.05 40	0.01 20	0.03 60	0.12 60	0.12 40	0.04 20	0.02 80	0.03 40
0.01 20	0.01 100	0.08 40	0.13 40	0.06 80	0.01 100	0.04 40	0.02 120
0.01 120	0.04 80	0.12 60	0.09 40	0.04 140	0.02 40	0.04 0	0.06 60

- Chuẩn hóa HOG bằng cách dùng block , chứa 2x2 cell hay 4x4 cell, các block có thể chồng lấn nhau, ta được vector đặc trưng cho khối. Nếu dùng block 2x2 cell thì vector đặc trưng có kích thước 36 vb=[v0, v1, v2, v3] gồm 4 vector của 4 cell đặt nối tiếp nhau. Tính chiều dài vector vb sau đó chia mỗi phần tử của vb cho chiều dài này để chuẩn hóa.
- Với ảnh n*m pixel và block 16*16, ta có tất cả (n-8)*(m-8)/64 khối, n và m là bội số của 8. Ảnh 128*64 có tất cả 105 khối hay 105 vector 36 chiều.
- Các vector này đưa vào SVM để phân lớp.
- SVM thường dùng để nhận dạng chữ số

https://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial/

HOG+SVM Pedestrian Detect

- Thư mục sources/samples/cpp có file peopledetect.cpp phát giác người đi bộ và letter_recog.cpp nhận dạng chữ. Đầu tiên ta khảo sát dạng đơn giản hóa của chương trình cpp.
- Khai báo bộ mô tả HOG:

cv::HOGDescriptor::HOGDescriptor(Size win_size=Size(64, 128), Size block_size=Size(16, 16), Size block_stride=Size(8, 8), Size cell_size=Size(8, 8), int nbins=9, double win_sigma=DEFAULT_WIN_SIGMA, double threshold_L2hys=0.2, bool gamma_correction=true, int nlevels=DEFAULT_NLEVELS).

DEFAULT_WIN_SIGMA = -1, DEFAULT_NLEVELS=64

Ví dụ HOGDescriptor hog; //dùng giá trị mặc định

Khai báo bộ phân lớp SVM

void setSVMDetector(const vector<float>& detector);

static vector<float> getDefaultPeopleDetector();

static vector<float> getPeopleDetector48x96();

static vector<float> getPeopleDetector64x128();

Ví dụ:
hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());

Tìm vị trí có người: nhận vào ảnh img, trả về hình chữ nhật bao quanh người đứng found_locations.

void detectMultiScale(const Mat& img, vector<Rect>& found_locations, double hit_threshold=0, Size win_stride=Size(), Size padding=Size(), double scale0=1.05,

int group_threshold=2);

Digit Classification Using HOG Features And SVM MATLAB Folder digitsvmHOGClassifier

- This example shows how to classify digits using HOG features and a multiclass SVM classifier.
- Object classification is an important task in many computer vision applications, including surveillance, automotive safety, and image retrieval. For example, in an automotive safety application, you may need to classify nearby objects as pedestrians or vehicles. Regardless of the type of object being classified, the basic procedure for creating an object classifier is:
- Acquire a labeled data set with images of the desired object.
- Partition the data set into a training set and a test set.
- Train the classifier using features extracted from the training set.
- Test the classifier using features extracted from the test set.

Digit Classification Using HOG Features And SVM MATLAB

- Data set is in folder matlab/toolbox/ vision.visiondata/ digit
- There are handwritten digit and synthetic digit, Each synthetic digit has 101 sample, total number is 1010 sample. Each handwritten digit has 12 sample. Each sample is a 16*16*3 image. Use synthetic sample to training and handwritten sample to test classifiers
- To illustrate, this example shows how to classify numerical digits using HOG (Histogram of Oriented Gradient) features and a multiclass SVM (Support Vector Machine) classifier. This type of classification is often used in many Optical Character Recognition (OCR) applications.
- The example uses the fitcecoc function from the Statistics and Machine Learning Toolbox™ and the extractHOGFeatures function from the Computer Vision System Toolbox™.

Digit Classification Using HOG Features And SVM MATLAB

```
% Load training and test data using |imageDatastore|.

syntheticDir   = fullfile(toolboxdir('vision'),
'visiondata','digits','synthetic');

handwrittenDir = fullfile(toolboxdir('vision'),
'visiondata','digits','handwritten');

% |imageDatastore| recursively scans the directory tree containing
the images. Folder names are automatically used as labels for each
image.

trainingSet = imageDatastore(syntheticDir,   'IncludeSubfolders',
true, 'LabelSource', 'foldernames');

testSet     = imageDatastore(handwrittenDir, 'IncludeSubfolders',
true, 'LabelSource', 'foldernames');

figure; subplot(1,2,1); imshow(trainingSet.Files{102});

subplot(1,2,2); imshow(testSet.Files{13});
```

Digit Classification Using HOG Features And SVM MATLAB

Digit Classification Using HOG Features And SVM MATLAB

Prior to training and testing a classifier, a pre-processing step is applied to remove noise artifacts introduced while collecting the image samples. This provides better feature vectors for training the classifier.

*% Show pre-processi*ng results

exTestImage = readimage(testSet,37);

processedImage = imbinarize(rgb2gray(exTestImage));

The data used to train the classifier are HOG feature vectors extracted from the training images. The extractHOGFeatures function returns a visualization output that can help form some intuition about just what the "right amount of information" means. By varying the HOG cell size parameter and visualizing the result, you can see the effect the cell size parameter has on the amount of shape information encoded in the feature vector:

Digit Classification Using HOG Features And SVM MATLAB

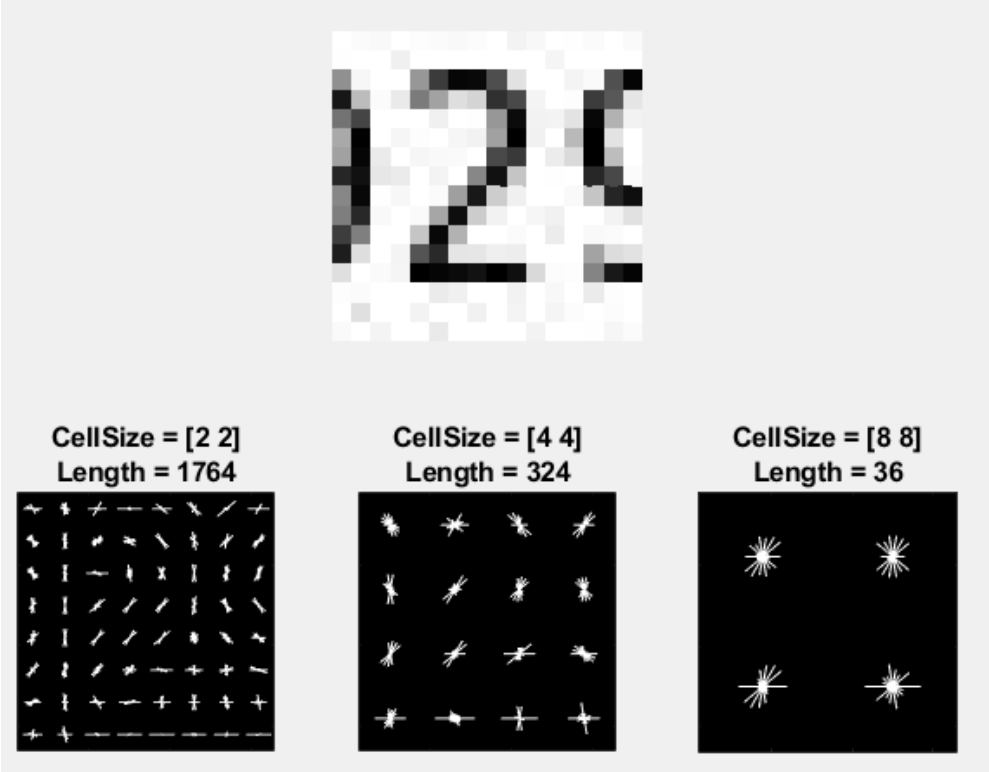
```
img = readimage(trainingSet, 206);
% Extract HOG features and HOG visualization
[hog_2x2, vis2x2] = extractHOGFeatures(img,'CellSize',[2 2]);
[hog_4x4, vis4x4] = extractHOGFeatures(img,'CellSize',[4 4]);
[hog_8x8, vis8x8] = extractHOGFeatures(img,'CellSize',[8 8]);
% Show the original image
figure;  subplot(2,3,1:3); imshow(img);
% Visualize the HOG features
subplot(2,3,4);   plot(vis2x2);
title({'CellSize = [2 2]'; ['Length = ' num2str(length(hog_2x2))]});
subplot(2,3,5); plot(vis4x4);
```

Digit Classification Using HOG Features And SVM MATLAB

```
title({'CellSize = [4
4]'; ['Length = '
num2str(length(hog_4
x4))]});

subplot(2,3,6);
plot(vis8x8);

title({'CellSize = [8
8]'; ['Length = '
num2str(length(hog_8
x8))]});
```



Digit Classification Using HOG Features And SVM MATLAB

The visualization shows that a cell size of [8 8] does not encode much shape information, while a cell size of [2 2] encodes a lot of shape information but increases the dimensionality of the HOG feature vector significantly. A good compromise is a 4-by-4 cell size. This size setting encodes enough spatial information to visually identify a digit shape while limiting the number of dimensions in the HOG feature vector, which helps speed up training. In practice, the HOG parameters should be varied with repeated classifier training and testing to identify the optimal parameter settings.

```
cellSize = [4 4];

hogFeatureSize = length(hog_4x4);
```

Digit Classification Using HOG Features And SVM MATLAB

Train a Digit Classifier. Digit classification is a multiclass classification problem, where you have to classify an image into one out of the ten possible digit classes. In this example, the fitcecoc function from the Statistics and Machine Learning Toolbox™ is used to create a multiclass classifier using binary SVMs.

Start by extracting HOG features from the training set. These features will be used to train the classifier.

% Loop over the trainingSet and extract HOG features from each image. A similar procedure will be used to extract features from the testSet.

```
numImages = numel(trainingSet.Files);

trainingFeatures = zeros(numImages, hogFeatureSize, 'single');

for i = 1:numImages

 img = readimage(trainingSet, i);
```

Digit Classification Using HOG Features And SVM MATLAB

```
img = rgb2gray(img);
 % Apply pre-processing steps
img = imbinarize(img);
trainingFeatures(i, :) = extractHOGFeatures(img, 'CellSize',
cellSize);
end
% Get labels for each image.
trainingLabels = trainingSet.Labels;
```

Next, train a classifier using the extracted features.

```
% fitcecoc uses SVM learners and a 'One-vs-One' encoding scheme.
classifier = fitcecoc(trainingFeatures, trainingLabels);
save(‘pathname’,’classifier’) % save to classifier.mat in pathname
```

Digit Classification Using HOG Features And SVM MATLAB

Evaluate the Digit Classifier

Evaluate the digit classifier using images from the test set, and generate a confusion matrix to quantify the classifier accuracy.

As in the training step, first extract HOG features from the test images. These features will be used to make predictions using the trained classifier.

% Extract HOG features from the test set. The procedure is similar to what was shown earlier and is encapsulated as a helper function for brevity.

[testFeatures, testLabels] = helperExtractHOGFeaturesFromImageSet(testSet, hogFeatureSize, cellSize);

% Make class predictions using the test features.

Digit Classification Using HOG Features And SVM MATLAB

```
predictedLabels = predict(classifier, testFeatures);

% Tabulate the results using a confusion matrix.

confMat = confusionmat(testLabels, predictedLabels);

helperDisplayConfusionMatrix(confMat)
```

The table shows the confusion matrix in percentage form. The columns of the matrix represent the predicted labels, while the rows represent the known labels. For this test set, digit 0 is often misclassified as 6, most likely due to their similar shapes. Similar errors are seen for 9 and 3. Training with a more representative data set like MNIST or SVHN , which contain thousands of handwritten characters, is likely to produce a better classifier compared with the one created using this synthetic data set.

After complete the test you can save classifier to your computer as classifier.mat in folder user/.. /Documents/matlab for after using

Digit Classification Using HOG Features And SVM MATLAB

digit	0	1	2	3	4	5	6	7	8	9
0	0.25	0.00	0.08	0.00	0.00	0.00	0.58	0.00	0.08	0.00
1	0.00	0.75	0.00	0.00	0.08	0.00	0.00	0.08	0.08	0.00
2	0.00	0.00	0.67	0.17	0.00	0.00	0.08	0.00	0.00	0.08
3	0.00	0.00	0.00	0.58	0.00	0.00	0.33	0.00	0.00	0.08
4	0.00	0.08	0.00	0.17	0.75	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.33	0.58	0.00	0.08	0.00
6	0.00	0.00	0.00	0.00	0.25	0.00	0.67	0.00	0.08	0.00
7	0.00	0.08	0.08	0.33	0.00	0.00	0.17	0.25	0.00	0.08
8	0.00	0.00	0.00	0.08	0.00	0.00	0.00	0.08	0.67	0.17
9	0.00	0.08	0.00	0.25	0.17	0.00	0.08	0.00	0.00	0.42

Digit Classification Using HOG Features And SVM MATLAB

Supporting Functions

```
function helperDisplayConfusionMatrix(confMat)
% Display the confusion matrix in a formatted table.
% Convert confusion matrix into percentage form
confMat = bsxfun(@rdivide,confMat,sum(confMat,2));
digits = '0':'9'; colHeadings =
arrayfun(@(x)sprintf('%d',x),0:9,'UniformOutput',false);
format = repmat('%-9s',1,11);
header = sprintf(format,'digit  |',colHeadings{:});
fprintf('\n%s\n%s\n',header,repmat('-',size(header)));
for idx = 1:numel(digits)
```

Digit Classification Using HOG Features And SVM MATLAB

```
 fprintf('%-9s',   [digits(idx) '      |']);
    fprintf('%-9.2f', confMat(idx,:));
    fprintf('\n')
end
End
function [features, setLabels] =
helperExtractHOGFeaturesFromImageSet(imds, hogFeatureSize,
cellSize)
% Extract HOG features from an imageDatastore.
setLabels = imds.Labels;
numImages = numel(imds.Files);
features  = zeros(numImages, hogFeatureSize, 'single');
```

Digit Classification Using HOG Features And SVM MATLAB

```
% Process each image and extract features
for j = 1:numImages
    img = readimage(imds, j);
    img = rgb2gray(img);
        % Apply pre-processing steps
    img = imbinarize(img);
features(j, :) = extractHOGFeatures(img,'CellSize',cellSize);
end
end
```

SVM MNIST digit classification in python using scikit-learn

https://github.com/ksopyla/svm_mnist_digit_classification

The project presents the well-known problem of MNIST handwritten digit classification The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

"SVM MNIST digit classification Sciklearn"

The MNIST database contains 60,000 training images and 10,000 testing images Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. There have been a number of scientific papers on attempts to achieve the lowest error rate; one paper, using a hierarchical system of convolutional neural networks, manages to get an error rate on the MNIST database of 0.23 percent. The original creators of the database keep a list of some of the methods tested on it. In their original paper, they use a support vector machine to get an error rate of 0.8 percent. An extended dataset similar to MNIST called EMNIST has been published in 2017, which contains 240,000 training images, and 40,000 testing images of handwritten digits and characters.

"SVM MNIST digit classification Sciklearn"

"SVM MNIST digit classification Sciklearn"

Project consist of three files:

mnist_helpers.py - contains some visualization functions: MNIST digits visualization and confusion matrix

svm_mnist_classification.py - script for SVM with RBF kernel classification

svm_mnist_embedings.py - script for linear SVM with embedings

SVM with RBF kernel

The svm_mnist_classification.py script downloads the MNIST database and visualizes some random digits. Next, it standardizes the data (mean=0, std=1) and launch grid search with cross-validation for finding the best parameters.

MNIST SVM kernel RBF Param search C=[0.1,0.5,1,5], gamma=[0.01,0.0.05,0.1,0.5].

"SVM MNIST digit classification Sciklearn"

mldata.org is a public repository for machine learning data, supported by the PASCAL network .

The sklearn.datasets package is able to directly download data sets from the repository using the function sklearn.datasets.fetch_mldata.

For example, to download the MNIST digit recognition database:

```
>>> from sklearn.datasets import fetch_mldata
>>> mnist = fetch_mldata('MNIST original', data_home=custom_data_home)
```

http://hanzratech.in/2015/02/24/handwritten-digit-recognition-using-opencv-sklearn-and-python.html

NHẬN DẠNG BIỂN SỐ XE

- LPR License Plate Recognition thường dùng ở các bãi xe, trạm thu phí giao thông, trạm cân xe...và camera giám sát giao thông.
- Biển số có hai loại hình chữ nhật hơi vuông dài bằng 1,3 chiều rông gắn sau xe và hình chữ nhật dài gấp 4 lần chiều rông gắn trước xe ô tô
- Có nhiều loại biển số xe cho xe máy và xe ô tô

- Nền biển màu xanh dương, chữ màu trắng là biển xe của các cơ quan hành chính sự nghiệp (dân sự)

- Nền biển màu trắng, chữ màu đen là xe thuộc sở hữu cá nhân và xe của các doanh nghiệp với 2 số đầu theo thứ tự các tỉnh, 4 hoặc 5 số cuối là số thứ tự cấp ngẫu nhiên.

- Nền biển màu đỏ, chữ màu trắng là dành riêng cho xe quân đội.

NHẬN DẠNG BIỂN SỐ XE

Riêng xe của các doanh nghiệp quân đội mang biển số 80 màu trắng. Bên cạnh đó, với biển số quân đội, 2 chữ cái đầu tiên là viết tắt của đơn vị cụ thể quản lý chiếc xe.

- Nền biển màu vàng chữ trắng là xe thuộc Bộ tư lệnh Biên phòng
- Nền biển màu vàng chữ đen là xe cơ giới chuyên dụng làm công trình

- Nền biển màu trắng với 2 chữ và năm số là biển cấp cho các đối tượng có yếu tố nước ngoài. Trong đó, biển NG là xe ngoại giao, biển NN là xe của các tổ chức, cá nhân nước ngoài. Trong 5 chữ số trên biển số, 3 số bên trái là mã quốc gia, 2 số tiếp theo là số thứ tự. Xe số 80 NG xxx-yy là biển cấp cho các đại sứ quán, thêm gạch đỏ ở giữa và 2 số cuối là 01 là biển xe của Tổng lãnh sự

NHẬN DẠNG BIỂN SỐ XE

NHẬN DẠNG BIỂN SỐ XE

NHẬN DẠNG BIỂN SỐ XE

- Nhận dạng biển số xe bao gồm các bước:

1/Tìm biển số: thường dùng thuật toán cascade classifier sau khi đã huấn luyện hoặc dùng contour

https://thigiacmaytinh.com/phat-hien-doi-tuong-p1-ly-thuyet/

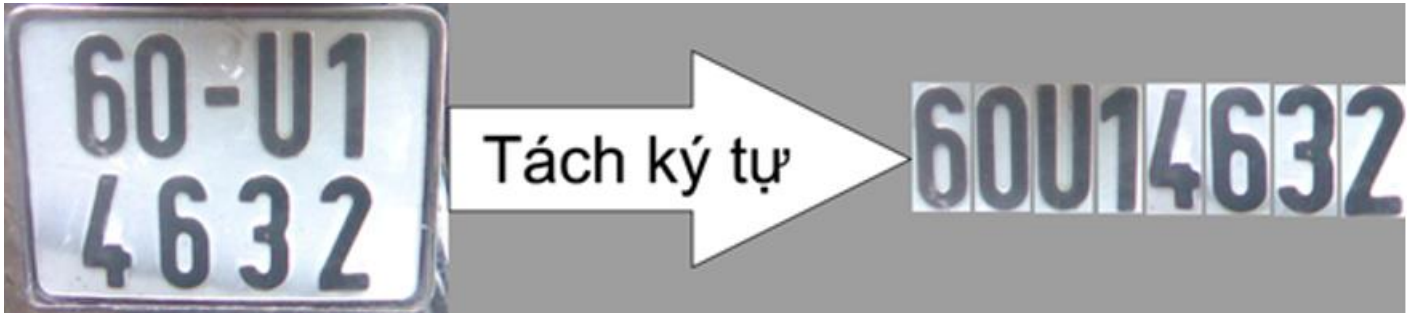
https://thigiacmaytinh.com/phat-hien-vat-the-p2-thuc-hanh/

NHẬN DẠNG BIỂN SỐ XE

2/ Lọc nhiễu, xoay ảnh nếu bị nghiêng

3/ Tách ký tự dùng thuật toán floodfill hay contour

https://thigiacmaytinh.com/su-dung-floodfill-de-tim-ky-tu/



4/ Nhận dạng ký tự dùng SVM, KNN...

https://thigiacmaytinh.com/ipss-phan-mem-doc-bien-so-xe-may/

https://thigiacmaytinh.com/nhan-dien-bien-xe-hoi-phan-13-tim-bien/