# 参考1.

# opencv支持向量机(python)

#### 支持向量机

支持向量机（Support Vector Machine, SVM）是一种二分类模型，目标是寻找一个标准（称为超平面）对样本数据进行分割，分割的原则是确保分类最优化（类别之间的间隔最大）。

当数据集较小时，使用支持向量机进行分类非常有效。

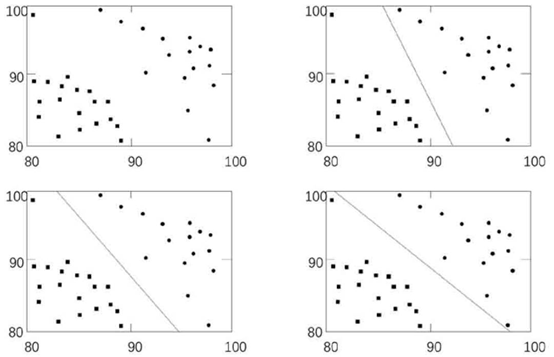
支持向量机是最好的现成分类器之一，“现成”是指分类器不加修改即可直接使用。

在对原始数据分类的过程中，可能无法使用线性方法实现分割。支持向量机在分类时，把无法线性分割的数据映射到高维空间，然后在高维空间找到分类最优的线性分类器。

Python支持向量机的库: sk-learn , LIBSVM等

OpenCV也提供了对支持向量机的支持

##### 理论基础

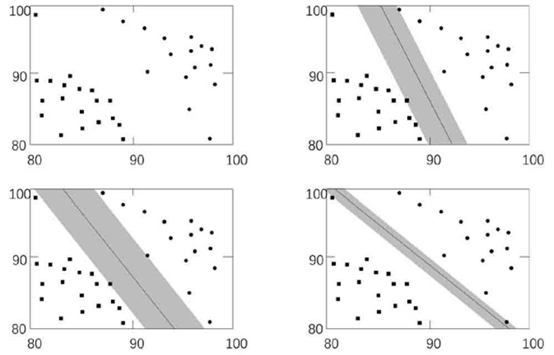


**用于划分不同类别的直线，就是分类器。**

构造分类器时，非常重要的一项工作就是找到最优分类器。

找到支持向量机：在已有数据中，找到离分类器最近的点，确保它们离分类器尽可能地远。

离分类器最近的点到分类器的距离称为间隔（margin）。希望间隔尽可能地大，这样分类器在处理数据时，就会更准确。

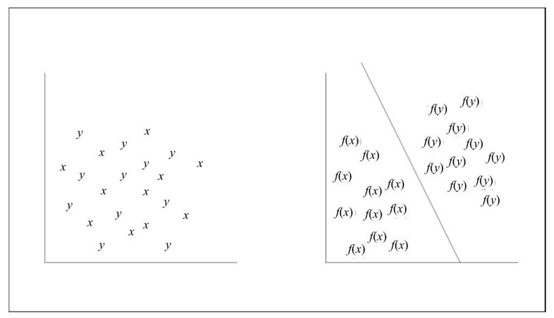


**离分类器最近的那些点叫作支持向量（support vector）。** 决定了分类器所在的位置。

*将不可分变为可分*

支持向量机会将不那么容易分类的数据通过函数映射变为可分类的。

支持向量机在处理数据时，如果在低维空间内无法完成分类，就会自动将数据映射到高维空间，使其变为（线性）可分的。简单地讲，就是对当前数据进行函数映射操作。



例如： 在分类时，通过函数f的映射，让左图中本来不能用线性分类器分类的数据变为右图中线性可分的数据。

同时: **支持向量机能够通过核函数有效地降低计算复杂度。**

实际上支持向量机可以处理任何维度的数据。在不同的维度下，支持向量机都会尽可能寻找类似于二维空间中的直线的线性分类器。

例如，在二维空间，支持向量机会寻找一条能够划分当前数据的直线；在三维空间，支持向量机会寻找一个能够划分当前数据的平面（plane）；在更高维的空间，支持向量机会尝试寻找一个能够划分当前数据的**超平面**（hyperplane）。

一般情况下，把能够可以被一条直线（更一般的情况，即一个超平面）分割的数据称为线性可分的数据，所以超平面是线性分类器。

**“支持向量机”是由“支持向量”和“机器”构成的。**

* “支持向量”是离分类器最近的那些点，这些点位于最大“间隔”上。通常情况下，分类仅依靠这些点完成，而与其他点无关。
* “机器”指的是分类器。

*支持向量机是一种基于关键点的分类算法。*

##### SVM使用介绍

在使用支持向量机模块时，需要先使用函数cv2.ml.SVM\_create()生成用于后续训练的空分类器模型。

语法格式：

svm = cv2.ml.SVM\_create()

获取了空分类器svm后，针对该模型使用svm.train()函数对训练数据进行训练

语法格式

训练结果= svm.train(训练数据，训练数据排列格式，训练数据的标签)

* 训练数据：表示原始数据，用来训练分类器。
* 训练数据排列格式：原始数据的排列形式有按行排列（cv2.ml.ROW\_SAMPLE，每一条训练数据占一行）和按列排列（cv2.ml.COL\_SAMPLE，每一条训练数据占一列）两种形式
* 训练数据的标签：原始数据的标签。
* 训练结果：训练结果的返回值。

例如： 用于训练的数据为data，其对应的标签为label，每一条数据按行排列，对分类器模型svm进行训练，所使用的语句为：

返回值 = svm.train(data, cv2.ml.ROW\_SAMPLE, label)

完成对分类器的训练后，使用svm.predict()函数即可使用训练好的分类器模型对测试数据进行分类，其语法格式为：

(返回值，返回结果) = svm.predict(测试数据)

*OpenCV支持对多个参数的自定义，例如：可以通过setType()函数设置类别，通过setKernel()函数设置核类型，通过setC()函数设置支持向量机的参数C ( 惩罚系数，即对误差的宽容度，默认值为0 ）。*

##### 例子介绍

题目： 已知员工的笔试成绩、面试成绩及对应的等级表现，根据新入职员工的笔试成绩、面试成绩预测其可能的表现。

首先构造一组随机数，并将其划分为两类，然后使用OpenCV自带的支持向量机模块完成训练和分类工作，最后将运算结果显示出来。

具体步骤：

1. 生成模拟数据

模拟生成入职一年后表现为A级的员工入职时的笔试和面试成绩。

构造20组笔试和面试成绩都分布在[95, 100)区间的数据对：

a = np.random.randint(95,100, (20, 2)).astype(np.float32)

上述模拟成绩，在一年后对应的工作表现为A级。

模拟生成入职一年后表现为B级的员工入职时的笔试和面试成绩。

构造20组笔试和面试成绩都分布在[90, 95)区间的数据对：

b = np.random.randint(90,95, (20, 2)).astype(np.float32)

上述模拟成绩，在一年后对应的工作表现为B级。

最后，将两组数据合并，并使用numpy.array对其进行类型转换：

data = np.vstack((a, b))

data = np.array(data, dtype='float32')

1. 构造分组标签

为对应表现为A级的分布在[95, 100)区间的数据，构造标签“0”：

aLabel=np.zeros((20,1))

为对应表现为B级的分布在[90, 95)区间的数据，构造标签“1”：

bLabel=np.ones((20,1))

将上述标签合并，并使用numpy.array对其进行类型转换：

label = np.vstack((aLabel, bLabel))

label = np.array(label, dtype='int32')

1. 训练

用支持向量机模块对已知的数据和其对应的标签进行训练：

svm = cv2.ml.SVM\_create()

result = svm.train(data, cv2.ml.ROW\_SAMPLE, label)

1. 分类

生成两个随机的数据对(笔试成绩，面试成绩)用于测试。

test = np.vstack([[98,90], [90,99]])

test = np.array(test, dtype='float32')

使用函数svm.predict()对随机成绩分类：

(p1, p2) = svm.predict(test)

1. 显示分类结果

将基础数据（训练数据）、用于测试的数据（测试数据）在图像上显示出来：

plt.scatter(a[:,0], a[:,1], 80, 'g', 'o')

plt.scatter(b[:,0], b[:,1], 80, 'b', 's')

plt.scatter(test[:,0], test[:,1], 80, 'r', '\*')

plt.show()

将测试数据及预测分类结果显示出来：

print(test)

print(p2)

**完整程序**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# 准备数据

a = np.random.randint(95,100, (20, 2)).astype(np.float32)

b = np.random.randint(90,95, (20, 2)).astype(np.float32)

data = np.vstack((a, b))

data = np.array(data, dtype='float32')

# 建立分组标签，0代表A级，1代表B级

aLabel=np.zeros((20,1))

bLabel=np.ones((20,1))

label = np.vstack((aLabel, bLabel))

label = np.array(label, dtype='int32')

# 训练

svm = cv2.ml.SVM\_create()

# 属性设置，直接采用默认值即可

#svm.setType(cv2.ml.SVM\_C\_SVC) # svm type

#svm.setKernel(cv2.ml.SVM\_LINEAR) # line

#svm.setC(0.01)

result = svm.train(data, cv2.ml.ROW\_SAMPLE, label)

#预测

test = np.vstack([[98,90], [90,99]])

test = np.array(test, dtype='float32')

(p1, p2) = svm.predict(test) # test 是 [[数据1],[数据2]] 结构的

# 结果

print(test)

print("res1",p2[0])

print("res2",p2[1])

plt.scatter(a[:,0], a[:,1], 80, 'g', 'o')

plt.scatter(b[:,0], b[:,1], 80, 'b', 's')

plt.scatter(test[:,0], test[:,1], 80, 'r', '\*')

plt.show()

效果：

|  |
| --- |
|  |
|  |
|  |

# 参考2

# OpenCV中的「SVM分类器」：基本原理、函数解析和示例代码

### 1. 引言

opencv中集成了基于libsvm[1](read://https_blog.csdn.net/?url=https%3A%2F%2Fblog.csdn.net%2FBit_Coders%2Farticle%2Fdetails%2F118493604#fn1)实现的SVM接口，便于直接进行视觉分类任务。

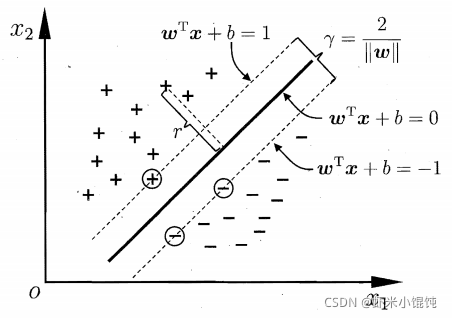
对于数据处理和可视化需求来说，可以用python接口opencv的SVM更加直观方便。

训练完模型后，将SVM模型保存为xml，可以在实时性应用中通过C++接口调用参数文件，进行实时推断。

在非均衡样本的分类训练中，用opencv中SVM默认的train函数，容易导致分类器偏向数量多的类别，这时可以采用trainAuto函数进行平衡。

如果你对SVM的原理有一定了解，可以直接跳转第3、4小节。

### 2. 基本原理

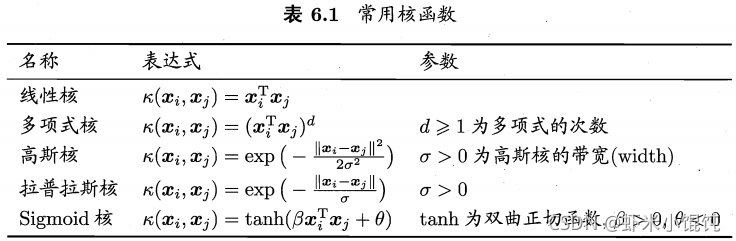


SVM旨在找到一个划分超平面，使得划分后的分类结果是最鲁棒的，对未见过的样本泛化性最好[2](read://https_blog.csdn.net/?url=https%3A%2F%2Fblog.csdn.net%2FBit_Coders%2Farticle%2Fdetails%2F118493604#fn2)。

在样本空间中，划分超平面可以用这个方程进行描述： w T x + b = 0 \boldsymbol{w}^T\boldsymbol{x}+b=0 wTx+b=0，其中 w = ( w 1 ; w 2 ; . . . ; w d ) \boldsymbol{w}=(w\_1;w\_2;...;w\_d) w=(w1​;w2​;...;wd​)为法向量，决定超平面的方向，b为位移项，决定超平面与原点之间的距离。

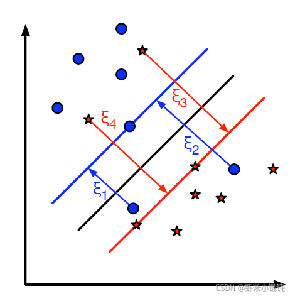
对于线性可分的样本空间，需要找到具有最大间隔（maximum margin）的划分超平面，即找到能使下式最大化的参数 w \boldsymbol{w} w和b[2](read://https_blog.csdn.net/?url=https%3A%2F%2Fblog.csdn.net%2FBit_Coders%2Farticle%2Fdetails%2F118493604#fn2)：  
min ⁡ w , b 1 2 ∣ ∣ w ∣ ∣ 2 \min\_{w,b}{\frac{1}{2}||\boldsymbol{w}||^2} w,bmin​21​∣∣w∣∣2s.t. y i ( w T x i + b ) ≥ 1 , i = 1 , 2 , . . . , m y\_i(\boldsymbol{w}^T\boldsymbol{x\_i}+b)≥1,i=1,2,...,m yi​(wTxi​+b)≥1,i=1,2,...,m

对于线性不可分的样本空间，可以将样本从原始空间映射到另一个**高维特征空间**，从而使样本在这个特征空间内线性可分。由于特征空间的维数可能很高，难以计算，所以通过引入核函数，可以将高维特征空间中的内积（dot product）转化为低维特征空间中的通过核函数计算的结果。

**常用核函数**[2](read://https_blog.csdn.net/?url=https%3A%2F%2Fblog.csdn.net%2FBit_Coders%2Farticle%2Fdetails%2F118493604#fn2)：  


为了减少过拟合，引入软间隔（soft margin）概念，允许支持向量机在一些样本上出错：  
y i ( w T x i + b ) ≥ 1 y\_i(\boldsymbol{w}^T\boldsymbol{x\_i}+b)≥1 yi​(wTxi​+b)≥1

用参数C来约束分类出错的样本，松弛变量 ξ i ξ\_i ξi​表示训练样本距离对应的正确决策边界的距离，对于分类正确的样本距离即为0[3](read://https_blog.csdn.net/?url=https%3A%2F%2Fblog.csdn.net%2FBit_Coders%2Farticle%2Fdetails%2F118493604#fn3)，所以实际累加的是出错样本的距离。

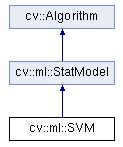


优化问题调整为：  
m i n w , b 0 ∣ ∣ w ∣ ∣ 2 + C ∑ i ξ i min\_{\boldsymbol{w},b\_0}{||\boldsymbol{w}||^2+C\sum\_i{ξ\_i}} minw,b0​​∣∣w∣∣2+Ci∑​ξi​

s.t. y i ( w T x i + b 0 ) ≥ 1 − ξ i , 且 ξ i ≥ 0 ∀ i y\_i(\boldsymbol{w}^T\boldsymbol{x\_i}+b\_0)≥1-ξ\_i,且ξ\_i≥0 ∀i yi​(wTxi​+b0​)≥1−ξi​,且ξi​≥0∀i

### 3. 函数解析

SVM类在opencv中的继承关系如图所示[4](read://https_blog.csdn.net/?url=https%3A%2F%2Fblog.csdn.net%2FBit_Coders%2Farticle%2Fdetails%2F118493604#fn4)：

  
SVM继承自StatModel和Algorithm类。

在opencv中使用SVM的一般流程如下：

训练

推理

开始

创建SVM模型

加载SVM模型

配置参数

加载训练数据

模型训练

保存模型

输入数据进行预测

#### 创建模型

C++：

static Ptr<SVM> cv::ml::SVM::create()

Python：

cv.ml.SVM\_create() -> retval

#### 设置模型类型

C++：

enum Types {

C\_SVC =100,//C-支持向量分类。n级分类（n≥ 2) 允许使用异常值的惩罚乘数 C 不完全地分离类。

NU\_SVC =101,//ν-支持向量分类。n级分类，可能有不完美的分离。参数ν用于代替C，参数ν在0-1范围内，值越大，决策边界越平滑。

ONE\_CLASS =102,//分布估计，所有的训练数据都来自同一个类，SVM 构建了一个边界，将类与特征空间的其余部分分开。

EPS\_SVR =103,//ε-支持向量回归。来自训练集的特征向量和拟合超平面之间的距离必须小于p。对于异常值，使用惩罚乘数 C。

NU\_SVR =104 // ν-支持向量回归。 ν用于代替 p。

}

virtual void cv::ml::SVM::setType(int val)

Python：

cv.ml\_SVM.setType(val) ->None

#### 设置参数C

根据"2.基本原理"中对参数C的介绍，我们应该如何设置参数C？

* C值较大时：误分类错误较少，但余量较小。这种情况下，侧重于寻找具有很少的误分类错误的超平面。
* C值较小时：具有更大余量和更多分类错误。在这种情况下，更侧重于寻找具有大余量的超平面。

C++:

//设置参数C

virtual void cv::ml::SVM::setC(double val)

python:

cv.ml\_SVM.setC(val) -> None

#### 设置核函数

C++:

enum KernelTypes {

CUSTOM =-1,//由SVM::getKernelType返回，默认是RBF

LINEAR =0,//线性内核，速度最快

POLY =1,//多项式核

RBF =2,//径向基函数（RBF），大多数情况下是个不错的选择

SIGMOID =3,//sigmoid核

CHI2 =4,//Chi2核，类似于RBF核

INTER =5//直方图交叉核，速度较快

}

virtual void cv::ml::SVM::setKernel(int kernelType)

python:

cv.ml\_SVM.setKernel(kernelType) -> None

#### 设置迭代算法的终止标准

C++:

virtual void cv::ml::SVM::setTermCriteria(const cv::TermCriteria &val)

// cv::TermCriteria

cv::TermCriteria::TermCriteria (int type,int maxCount,double epsilon)

// Type

enum cv::TermCriteria::Type {

COUNT =1,

MAX\_ITER =COUNT,//最大迭代次数

EPS =2 //迭代算法停止时所需的精度或参数变化

}

python:

cv.ml\_SVM.setTermCriteria(val) ->None

#### 训练SVM模型

trainAuto方法通过选择最佳参数 C、gamma、p、nu、coef0、degree 来自动训练 SVM 模型。当测试集误差的交叉验证估计最小时，参数被认为是最佳的。此函数仅使用SVM::getDefaultGrid进行参数优化，因此仅提供基本的参数选项。

trainAuto函数适用于分类（SVM::C\_SVC或SVM::NU\_SVC）以及回归（SVM::EPS\_SVR或SVM::NU\_SVR）。如果是SVM::ONE\_CLASS，则不进行优化，并执行带有 params 中指定参数的常用 SVM。

C++：

//输入由TrainData::create或TrainData::loadFromCSV构造的训练数据

virtual bool cv::ml::SVM::trainAuto(const Ptr<TrainData> & data,

int kFold = 10,

ParamGrid Cgrid = getDefaultGrid(C),

ParamGrid gammaGrid = getDefaultGrid(GAMMA),

ParamGrid pGrid = getDefaultGrid(P),

ParamGrid nuGrid = getDefaultGrid(NU),

ParamGrid coeffGrid = getDefaultGrid(COEF),

ParamGrid degreeGrid = getDefaultGrid(DEGREE),

bool balanced = false

)

//输入训练样本

bool cv::ml::SVM::trainAuto(InputArray samples,

int layout,

InputArray responses,

int kFold = 10,

Ptr< ParamGrid > Cgrid = SVM::getDefaultGridPtr(SVM::C),

Ptr< ParamGrid > gammaGrid = SVM::getDefaultGridPtr(SVM::GAMMA),

Ptr< ParamGrid > pGrid = SVM::getDefaultGridPtr(SVM::P),

Ptr< ParamGrid > nuGrid = SVM::getDefaultGridPtr(SVM::NU),

Ptr< ParamGrid > coeffGrid = SVM::getDefaultGridPtr(SVM::COEF),

Ptr< ParamGrid > degreeGrid = SVM::getDefaultGridPtr(SVM::DEGREE),

bool balanced = false

)

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

Python：

cv.ml\_SVM.trainAuto(samples, layout, responses[, kFold[, Cgrid[, gammaGrid[, pGrid[, nuGrid[, coeffGrid[, degreeGrid[, balanced]]]]]]]]) -> retval

参数：

* samples：训练样本
* layout：参考 ml::SampleTypes，如cv.ml.ROW\_SAMPLE表示每个训练样本是行向量，cv.ml.COL\_SAMPLE表示每个训练样本是列向量
* responses：与训练样本有关的响应向量
* kFold：k交叉验证，训练集会分成k个子集，从中选取一个用来测试，剩余k-1个用来训练
* balanced：如果设为True且是2-class分类问题，方法会自动创建更平衡的交叉验证子集，即子集中的类之间比例接近整个训练数据集中的比例

#### 预测结果

C++：

// 预测输入样本的响应结果

virtual float predict(

InputArray samples, // input samples, float matrix

OutputArray results = cv::noArray(), // optional output results matrix

int flags = 0 // (model-dependent)

) const = 0;

python：

cv.ml\_StatModel.predict(samples[, results[, flags]]) ->retval, results

#### 误差计算

对于回归模型，误差计算为 RMS；对于分类器，误差计算为错误分类样本的百分比 (0%-100%)。  
C++：

// 在训练集或测试集上计算误差

virtual float calcError(

const Ptr<TrainData>& data, // training samples

bool test, // true: compute over test set

// false: compute over training set

cv::OutputArray resp // the optional output responses

) const;

python：

cv.ml\_StatModel.calcError(data, test[, resp]) ->retval, resp

#### 保存SVM模型

C++：

void cv::Algorithm::save(const String &filename) const

Python：

cv.Algorithm.save(filename) ->None

#### 从文件中加载SVM

C++：

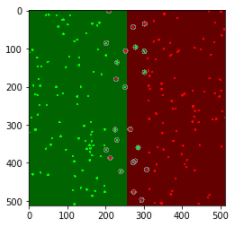
static Ptr<SVM> cv::ml::SVM::load(const String &filepath)

Python：

cv.ml.SVM\_load(filepath) ->retval

### 4. 示例代码

#### 官方示例（python）

  
构造数据，用来模拟训练集中的两个类别：

from \_\_future\_\_ import print\_function

import cv2 as cv

import numpy as np

import random as rng

import time

from matplotlib import pyplot as plt

NTRAINING\_SAMPLES = 100 # 每个类别的训练样本数

FRAC\_LINEAR\_SEP = 0.9 # 线性可分的样本比例

# 准备用于数据可视化

WIDTH = 512

HEIGHT = 512

I = np.zeros((HEIGHT, WIDTH, 3), dtype=np.uint8)

# 设置训练样本

trainData = np.empty((2\*NTRAINING\_SAMPLES, 2), dtype=np.float32)

labels = np.empty((2\*NTRAINING\_SAMPLES, 1), dtype=np.int32)

rng.seed(100) # Random value generation class

# 线性可分的训练样本数量

nLinearSamples = int(FRAC\_LINEAR\_SEP \* NTRAINING\_SAMPLES)

## [setup1]

# 生成class 1的随机点，随机点的x坐标在[0, 0.4)，y坐标在 [0, 1)

trainClass = trainData[0:nLinearSamples,:]

# The x coordinate of the points is in [0, 0.4)

c = trainClass[:,0:1]

c[:] = np.random.uniform(0.0, 0.4 \* WIDTH, c.shape)

# The y coordinate of the points is in [0, 1)

c = trainClass[:,1:2]

c[:] = np.random.uniform(0.0, HEIGHT, c.shape)

# 生成class 2的随机点，随机点的x坐标在[0.6, 1]，y坐标在 [0, 1)

trainClass = trainData[2\*NTRAINING\_SAMPLES-nLinearSamples:2\*NTRAINING\_SAMPLES,:]

# The x coordinate of the points is in [0.6, 1]

c = trainClass[:,0:1]

c[:] = np.random.uniform(0.6\*WIDTH, WIDTH, c.shape)

# The y coordinate of the points is in [0, 1)

c = trainClass[:,1:2]

c[:] = np.random.uniform(0.0, HEIGHT, c.shape)

# 设置线性不可分的训练样本

# Generate random points for the classes 1 and 2

trainClass = trainData[nLinearSamples:2\*NTRAINING\_SAMPLES-nLinearSamples,:]

# x坐标在 [0.4, 0.6)，y坐标在[0, 1)

c = trainClass[:,0:1]

c[:] = np.random.uniform(0.4\*WIDTH, 0.6\*WIDTH, c.shape)

c = trainClass[:,1:2]

c[:] = np.random.uniform(0.0, HEIGHT, c.shape)

# 设置两个类别的label

labels[0:NTRAINING\_SAMPLES,:] = 1 # Class 1

labels[NTRAINING\_SAMPLES:2\*NTRAINING\_SAMPLES,:] = 2 # Class 2

设置SVM参数，初始化模型：

print('Starting training process')

svm = cv.ml.SVM\_create()

svm.setType(cv.ml.SVM\_C\_SVC)

svm.setC(0.1)

svm.setKernel(cv.ml.SVM\_LINEAR)

svm.setTermCriteria((cv.TERM\_CRITERIA\_MAX\_ITER, int(1e7), 1e-6))

训练SVM：

## 训练

svm.train(trainData, cv.ml.ROW\_SAMPLE, labels)

print('Finished training process')

## 显示决策区域

green = (0,100,0)

blue = (100,0,0)

for i in range(I.shape[0]):

for j in range(I.shape[1]):

sampleMat = np.matrix([[j,i]], dtype=np.float32)

response = svm.predict(sampleMat)[1]

if response == 1:

I[i,j] = green

elif response == 2:

I[i,j] = blue

对训练集中两个类别的样本进行可视化：

## 用两种颜色圆圈表示class 1和class 2的训练数据

thick = -1

# Class 1

for i in range(NTRAINING\_SAMPLES):

px = trainData[i,0]

py = trainData[i,1]

cv.circle(I, (px, py), 3, (0, 255, 0), thick)

# Class 2

for i in range(NTRAINING\_SAMPLES, 2\*NTRAINING\_SAMPLES):

px = trainData[i,0]

py = trainData[i,1]

cv.circle(I, (px, py), 3, (255, 0, 0), thick)

# 显示支持向量（

## [show\_vectors]

thick = 2

sv = svm.getUncompressedSupportVectors()

for i in range(sv.shape[0]):

cv.circle(I, (sv[i,0], sv[i,1]), 6, (128, 128, 128), thick)

## [show\_vectors]

#cv.imwrite('result.png', I) # save the Image

#cv.imshow('SVM for Non-Linear Training Data', I) # show it to the user

plt.imshow(I)

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

#### 推理阶段（C++版本）

void test\_svm(std::string videopath, std::string svm\_file = "svm.mat")

{

/// 加载svm模型参数

cv::Ptr<cv::ml::SVM> svm = cv::ml::SVM::load(svm\_file);

/// 初始化特征提取器

// 此处省略……

cv::VideoCapture cap(videopath);

if (cap.isOpened())

{

cv::Mat src;//img

int sleep\_interval = 1;//每隔多少ms取帧

int frameIdx = 0;

while (true)

{

if (!cap.read(src))

{

break;

}

frameIdx++;

double start = static\_cast<double>(cv::getTickCount());

cv::Mat flowFeat;

//提取运动特征

m\_featureExtactor.ProcessFlow(src, flowFeat);

flowFeat.convertTo(flowFeat, CV\_32FC1);

//获取分类结果

int response = (int)svm->predict(flowFeat);

cv::putText(src, cv::String(std::to\_string(response)), cv::Point(20,20), cv::FONT\_HERSHEY\_PLAIN, 1, cv::Scalar(0, 255, 0));

//计算耗时

float times = ((float)cv::getTickCount() - start) / cv::getTickFrequency();

std::cout << "time cost: " << times << " s." << std::endl;

cv::imshow("img", src);

if (cv::waitKey(1) == 27) {

break;

}

}

}

}

### 5. 小结

本文整理了Opencv中SVM支持向量机的原理、函数和代码示例。

如果对你有帮助的话，欢迎一键三连支持下博主~

1. Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST), 2(3):27, 2011. [↩︎](read://https_blog.csdn.net/?url=https%3A%2F%2Fblog.csdn.net%2FBit_Coders%2Farticle%2Fdetails%2F118493604#fnref1)
2. 《机器学习》周志华 [↩︎](read://https_blog.csdn.net/?url=https%3A%2F%2Fblog.csdn.net%2FBit_Coders%2Farticle%2Fdetails%2F118493604#fnref2) [↩︎](read://https_blog.csdn.net/?url=https%3A%2F%2Fblog.csdn.net%2FBit_Coders%2Farticle%2Fdetails%2F118493604#fnref2:1) [↩︎](read://https_blog.csdn.net/?url=https%3A%2F%2Fblog.csdn.net%2FBit_Coders%2Farticle%2Fdetails%2F118493604#fnref2:2)
3. https://docs.opencv.org/4.5.3/d4/db1/tutorial\_py\_svm\_basics.html [↩︎](read://https_blog.csdn.net/?url=https%3A%2F%2Fblog.csdn.net%2FBit_Coders%2Farticle%2Fdetails%2F118493604#fnref3)
4. https://docs.opencv.org/4.5.3/d1/d2d/classcv\_1\_1ml\_1\_1SVM.html [↩︎](read://https_blog.csdn.net/?url=https%3A%2F%2Fblog.csdn.net%2FBit_Coders%2Farticle%2Fdetails%2F118493604#fnref4)

# 参考3

# Python，OpenCV基于支持向量机SVM的手写数字OCR

[上一节介绍了基于KNN的手写数字OCR+字母OCR](https://link.juejin.cn/?target=https%3A%2F%2Fblog.csdn.net%2Fqq_40985985%2Farticle%2Fdetails%2F119419147)，这一节将介绍基于支持向量机SVM的手写数字OCR。

## 1. 效果图

**简单线性向量机训练效果图如下：** 图中有4个点，3个趋于白色点，一个灰黑色点，可以看到分割线的决策边界很明显。

**非线性向量机训练数据效果图如下：**

下图中绿色、蓝色点杂糅在一起，中间的决策边界是非线性的，但可以近似为线性。边界有灰色圆圈的点是 **支持向量**，依赖这些少量的数据就可以找到 **决策边界**。

## 2. SVM及原理

*支持向量机SVM（Supported Vector Machines)*

要了解SVM，首先需要了解线性可分数据及线性不可分数据，简单来说，就是在平面或多维有一堆点进行分类，能否用一根线分隔以分类彼此。

* 线性可分数据

**KNN需要计算测试数据到所有点的距离，当数据量比较大的时候，需要较大的内存来存储。** 可以有另一种思路：找到一条线 f(x)=ax\_1+bx\_2+c ，它将数据分为两个区域。当得到一个新的 test\_data X 时，只需将其替换为 f(x)。如果 f(X) > 0，则属于蓝色组，否则属于红色组。

称这条线为 **决策边界，它非常简单且节省内存。** 这种可以用直线（或更高维的超平面）一分为二的数据称为 **线性可分数据**。

* 低维空间中的非线性可分离数据在高维空间中变为线性可分离的可能性更大。在上图中可以看到很多这样的线条是可能的。**要拿哪一个？非常直观，可以说这条线应该尽可能远离所有点。**

**走最远的线路将提供更多的抗噪能力。所以SVM所做的就是找到一条到训练样本最小距离最大的直线（或超平面）。**

* 要找到这个决策边界，并不需要所有数据，只需要那些靠近相反群体的数据。

在该图像中，它们是一个蓝色实心圆圈和两个红色实心方块。我们可以称它们为支持向量，穿过它们的线称为**支持平面**。它们足以找到**决策边界**。

* **权重向量决定决策边界的方向，而偏置点决定其位置。**

## 2. 源码

### 2.1 SVM的手写数字OCR

# 使用SVM进行手写数据OCR

# 在KNN中直接使用像素强度作为特征向量。

# 在SVM中使用方向梯度直方图（HOG Histogram of Oriented Gradients）作为特征向量。

# 在这里，使用二阶矩对图像进行反扭曲。

import cv2

import numpy as np

SZ = 20

bin\_n = 16 # Number of bins

svm\_params = dict(kernel\_type=cv2.ml.SVM\_LINEAR,

svm\_type=cv2.ml.SVM\_C\_SVC,

C=2.67, gamma=5.383)

affine\_flags = cv2.WARP\_INVERSE\_MAP | cv2.INTER\_LINEAR

# 左图像是原始图像，右图像是倾斜图像。

def deskew(img):

m = cv2.moments(img)

if abs(m['mu02']) < 1e-2:

return img.copy()

skew = m['mu11'] / m['mu02']

M = np.float32([[1, skew, -0.5 \* SZ \* skew], [0, 1, 0]])

img = cv2.warpAffine(img, M, (SZ, SZ), flags=affine\_flags)

return img

# （HOG Histogram of Oriented Gradients）方向梯度直方图

def hog(img):

gx = cv2.Sobel(img, cv2.CV\_32F, 1, 0)

gy = cv2.Sobel(img, cv2.CV\_32F, 0, 1)

mag, ang = cv2.cartToPolar(gx, gy)

# 量化 (0...16)的binvalues

bins = np.int32(bin\_n \* ang / (2 \* np.pi))

# 分成四个子块

bin\_cells = bins[:10, :10], bins[10:, :10], bins[:10, 10:], bins[10:, 10:]

mag\_cells = mag[:10, :10], mag[10:, :10], mag[:10, 10:], mag[10:, 10:]

hists = [np.bincount(b.ravel(), m.ravel(), bin\_n) for b, m in zip(bin\_cells, mag\_cells)]

hist = np.hstack(hists)

return hist

img = cv2.imread('images/digits.png', 0)

print(img.shape) # (1000,2000)

cells = [np.hsplit(row, 100) for row in np.vsplit(img, 50)]

print(len(cells)) # 50\*100

# 一半数据用于训练，一半用于测试(前50列，后50列）

train\_cells = [i[:50] for i in cells]

test\_cells = [i[50:] for i in cells]

# cv2.imshow("img", train\_cells[0][0])

# cv2.imshow("deskew", deskew(train\_cells[0][0]))

# cv2.waitKey(0)

# 训练数据

deskewed = [list(map(deskew, row)) for row in train\_cells]

hogdata = [list(map(hog, row)) for row in deskewed]

trainData = np.float32(hogdata).reshape(-1, 64)

responses = np.repeat(np.arange(10), 250)[:, np.newaxis]

print('trainData: ', type(trainData), len(trainData))

print('responses: ', type(responses), responses.shape, len(responses))

print(responses[0])

svm = cv2.ml.SVM\_create()

svm.setGamma(svm\_params['gamma'])

svm.setC(svm\_params['C'])

svm.setKernel(cv2.ml.SVM\_LINEAR)

svm.setType(cv2.ml.SVM\_C\_SVC)

svm.train(trainData, cv2.ml.ROW\_SAMPLE, responses)

# 把训练的数据及模型保存下来

svm.save('images/svm\_data.dat')

# 测试数据

deskewed = [list(map(deskew, row)) for row in test\_cells]

hogdata = [list(map(hog, row)) for row in deskewed]

testData = np.float32(hogdata).reshape(-1, bin\_n \* 4)

result = svm.predict(testData)[1]

print('result: ', type(result))

print('responses: ', type(responses))

# 检查准确度

mask = result == responses

correct = np.count\_nonzero(mask)

print('correct: ', correct)

# SVM得到93.8%的准确度，比KNN的91.76%要高一些

print(correct \* 100.0 / len(list(result)))

### 2.2 非线性SVM

from \_\_future\_\_ import print\_function

import random as rng

import cv2 as cv

import numpy as np

NTRAINING\_SAMPLES = 100 # Number of training samples per class

FRAC\_LINEAR\_SEP = 0.9 # Fraction of samples which compose the linear separable part

# 可视化窗口大小

WIDTH = 512

HEIGHT = 512

I = np.zeros((HEIGHT, WIDTH, 3), dtype=np.uint8)

# 随机生成训练数据

trainData = np.empty((2 \* NTRAINING\_SAMPLES, 2), dtype=np.float32)

labels = np.empty((2 \* NTRAINING\_SAMPLES, 1), dtype=np.int32)

rng.seed(100) # 随机生成分类标签

# 为训练数据设置线性分离区

# Set up the linearly separable part of the training data

nLinearSamples = int(FRAC\_LINEAR\_SEP \* NTRAINING\_SAMPLES)

# 为分类1生成随机点

trainClass = trainData[0:nLinearSamples, :]

# x在[0,0.4]

c = trainClass[:, 0:1]

c[:] = np.random.uniform(0.0, 0.4 \* WIDTH, c.shape)

# y在[0, 1)

c = trainClass[:, 1:2]

c[:] = np.random.uniform(0.0, HEIGHT, c.shape)

# 为分类2生成随机点

trainClass = trainData[2 \* NTRAINING\_SAMPLES - nLinearSamples:2 \* NTRAINING\_SAMPLES, :]

# x在 [0.6, 1]

c = trainClass[:, 0:1]

c[:] = np.random.uniform(0.6 \* WIDTH, WIDTH, c.shape)

# y在 [0, 1)

c = trainClass[:, 1:2]

c[:] = np.random.uniform(0.0, HEIGHT, c.shape)

# 为测试数据集的分类1，2分别生成随机点

trainClass = trainData[nLinearSamples:2 \* NTRAINING\_SAMPLES - nLinearSamples, :]

# x在[0.4,0.6]

c = trainClass[:, 0:1]

c[:] = np.random.uniform(0.4 \* WIDTH, 0.6 \* WIDTH, c.shape)

# y在[0,1]

c = trainClass[:, 1:2]

c[:] = np.random.uniform(0.0, HEIGHT, c.shape)

# 设置分类标签1及2

labels[0:NTRAINING\_SAMPLES, :] = 1 # 分类1

labels[NTRAINING\_SAMPLES:2 \* NTRAINING\_SAMPLES, :] = 2 # 分类2

# 开始训练，首先设置支持向量机SVM参数

print('Starting training process')

# 初始化

svm = cv.ml.SVM\_create()

svm.setType(cv.ml.SVM\_C\_SVC)

svm.setC(0.1)

svm.setKernel(cv.ml.SVM\_LINEAR)

svm.setTermCriteria((cv.TERM\_CRITERIA\_MAX\_ITER, int(1e7), 1e-6))

# 训练SVM

svm.train(trainData, cv.ml.ROW\_SAMPLE, labels)

# 结束训练

print('Finished training process')

# 展示决策区域（绘制蓝色，绿色） 分类1为绿色，分类2为蓝色

green = (0, 100, 0)

blue = (100, 0, 0)

for i in range(I.shape[0]):

for j in range(I.shape[1]):

sampleMat = np.matrix([[j, i]], dtype=np.float32)

response = svm.predict(sampleMat)[1]

if response == 1:

I[i, j] = green

elif response == 2:

I[i, j] = blue

# 展示测试数据

thick = -1

# 分类1 绿色

for i in range(NTRAINING\_SAMPLES):

px = trainData[i, 0]

py = trainData[i, 1]

cv.circle(I, (px, py), 3, (0, 255, 0), thick)

# 分类2 蓝色

for i in range(NTRAINING\_SAMPLES, 2 \* NTRAINING\_SAMPLES):

px = trainData[i, 0]

py = trainData[i, 1]

cv.circle(I, (px, py), 3, (255, 0, 0), thick)

# 展示支持向量

thick = 2

sv = svm.getUncompressedSupportVectors()

for i in range(sv.shape[0]):

cv.circle(I, (sv[i, 0], sv[i, 1]), 6, (128, 128, 128), thick)

cv.imwrite('non\_linear\_svms\_result.png', I) # 保存图片

cv.imshow('SVM for Non-Linear Training Data', I) # 展示图片结果

cv.waitKey()

### 效果：

|  |
| --- |
| 在这里插入图片描述 |