SVM的图像分类算法实现

Author: WENG Email:hurricanblue@126.com Date: 2014-12-31

# **SVM LIB程序包**

LIBSVM是台湾大学林智仁(Chi h-Jen Lin)博士等开发设计的一个操作简单、易于使用、快速有效的通用SVM软件包，可以解决分类问题（包括C-SVC、n-SVC）、回归问题（包括e-SVR、n-SVR）以及分布估计（one-class-SVM）等问题，提供了线性、多项式、径向基和S形函数四种常用的核函数供选择，可以有效地解决多类问题、交叉验证选择参数、对不平衡样本加权、多类问题的概率估计等。LIBSVM是一个开源的软件包，需要者都可以免费的从作者的[个人主页](http://www.csie.ntu.edu.tw/~cjlin/)处获得。他不仅提供了LIBSVM的C++语言的算法源代码，还提供了Python、Java、R、MATLAB、Perl、Ruby、LabVIEW以及C#.net等各种语言的接口，可以方便的在Windows或UNIX平台下使用，也便于科研工作者根据自己的需要进行改进（譬如设计使用符合自己特定问题需要的核函数等）。另外还提供了WINDOWS平台下的可视化操作工具SVM-toy，并且在进行模型参数选择时可以绘制出交叉验证精度的等高线图。

[SVM算法简介](file:///F:\JOBS\SVM_MRS_OBC(2014.10.9)\SVM%20资料) (文件夹)

# LIB库

## Download

<http://pan.baidu.com/s/1bn6fhD1> （libsvm-3.20.zip）

<http://www.csie.ntu.edu.tw/~cjlin/libsvm/index.html?js=1#download> （SVM发布主页）

## Readme documents (部分)

[All the details is in the readme file](LIB/libsvm-3.20/matlab/README)

Examples

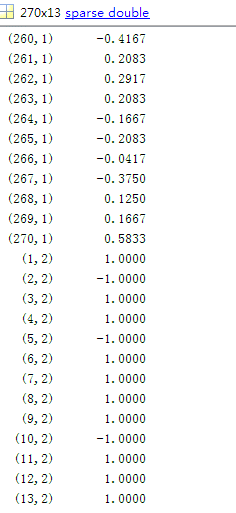
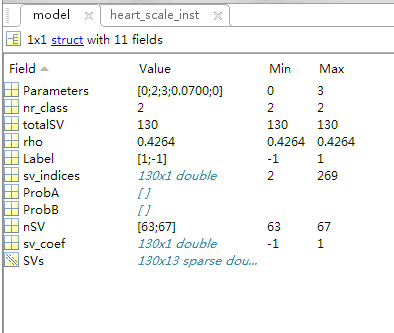
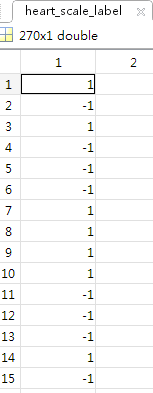
========

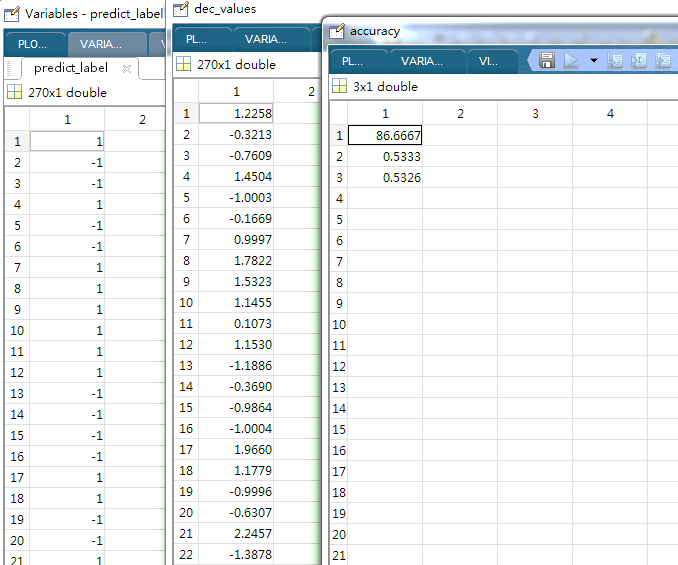
（1）Train and test on the provided data heart\_scale:

matlab> [heart\_scale\_label, heart\_scale\_inst] = libsvmread('../heart\_scale');

matlab> model = svmtrain(heart\_scale\_label, heart\_scale\_inst, '-c 1 -g 0.07');

matlab> [predict\_label, accuracy, dec\_values] = svmpredict(heart\_scale\_label, heart\_scale\_inst, model); % test the training data



（2）For probability estimates, you need '-b 1' for training and testing:

matlab> [heart\_scale\_label, heart\_scale\_inst] = libsvmread('../heart\_scale');

matlab> model = svmtrain(heart\_scale\_label, heart\_scale\_inst, '-c 1 -g 0.07 -b 1');

matlab> [heart\_scale\_label, heart\_scale\_inst] = libsvmread('../heart\_scale');

matlab> [predict\_label, accuracy, prob\_estimates] = svmpredict(heart\_scale\_label, heart\_scale\_inst, model, '-b 1');

（3）To use precomputed kernel, you must include sample serial number as the first column of the training and testing data (assume your kernel matrix is K, # of instances is n):

matlab> K1 = [(1:n)', K]; % include sample serial number as first column

matlab> model = svmtrain(label\_vector, K1, '-t 4');

matlab> [predict\_label, accuracy, dec\_values] = svmpredict(label\_vector, K1, model); % test the training data

（4）We give the following detailed example by splitting heart\_scale into 150 training and 120 testing data. Constructing a linear kernel matrix and then using the precomputed kernel gives exactly the same testing error as using the LIBSVM built-in linear kernel.

matlab> [heart\_scale\_label, heart\_scale\_inst] = libsvmread('../heart\_scale');

matlab>

matlab> % Split Data

matlab> train\_data = heart\_scale\_inst(1:150,:);

matlab> train\_label = heart\_scale\_label(1:150,:);

matlab> test\_data = heart\_scale\_inst(151:270,:);

matlab> test\_label = heart\_scale\_label(151:270,:);

matlab>

matlab> % Linear Kernel

matlab> model\_linear = svmtrain(train\_label, train\_data, '-t 0');

matlab> [predict\_label\_L, accuracy\_L, dec\_values\_L] = svmpredict(test\_label, test\_data, model\_linear);

matlab>

matlab> % Precomputed Kernel

matlab> model\_precomputed = svmtrain(train\_label, [(1:150)', train\_data\*train\_data'], '-t 4');

matlab> [predict\_label\_P, accuracy\_P, dec\_values\_P] = svmpredict(test\_label, [(1:120)', test\_data\*train\_data'], model\_precomputed);

matlab>

matlab> accuracy\_L % Display the accuracy using linear kernel

matlab> accuracy\_P % Display the accuracy using precomputed kernel

Note that for testing, you can put anything in the testing\_label\_vector. For more details of precomputed kernels, please read the section ``Precomputed Kernels'' in the README of the LIBSVM package.

(5) Examples of options: -s 0 -c 10 -t 1 -g 1 -r 1 -d 3

Classify a binary data with polynomial kernel (u'v+1)^3 and C = 10

options:

-s svm\_type : set type of SVM (default 0)

0 -- C-SVC (multi-class classification)

1 -- nu-SVC (multi-class classification)

2 -- one-class SVM

3 -- epsilon-SVR (regression)

4 -- nu-SVR (regression)

-t kernel\_type : set type of kernel function (default 2)

0 -- linear: u'\*v

1 -- polynomial: (gamma\*u'\*v + coef0)^degree

2 -- radial basis function: exp(-gamma\*|u-v|^2)

3 -- sigmoid: tanh(gamma\*u'\*v + coef0)

-d degree : set degree in kernel function (default 3)

-g gamma : set gamma in kernel function (default 1/num\_features)

-r coef0 : set coef0 in kernel function (default 0)

-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)

-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)

-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)

-m cachesize : set cache memory size in MB (default 100)

-e epsilon : set tolerance of termination criterion (default 0.001)

-h shrinking: whether to use the shrinking heuristics, 0 or 1 (default 1)

-b probability\_estimates: whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)

-wi weight: set the parameter C of class i to weight\*C, for C-SVC (default 1)

The k in the -g option means the number of attributes in the input data.

（6）[LIBSVM使s用方法](http://blog.csdn.net/kobesdu/article/details/8944851)

LibSVM是以源代码和.exe可执行文件两种方式给出的。如果是Windows系列操作系统，可以直接使用软件包提供的程序，也可以进行修改编译；如果是Unix类系统，必须自己编译，软件包中提供了编译格式文件，个人在SGI工作站(操作系统IRIX6.5)上，使用免费编译器GNU C++3.3编译通过。

LIBSVM 使用的一般步骤是：

1） 按照LIBSVM软件包所要求的格式准备数据集；

2） 对数据进行简单的缩放操作；

3） 考虑选用RBF 核函数；

4） 采用交叉验证选择最佳参数C与g ；

5） 采用最佳参数C与g 对整个训练集进行训练获取支持向量机模型；

6） 利用获取的模型进行测试与预测。

**LIBSVM使用的数据格式**

该软件使用的训练数据和检验数据文件格式如下：



<label> 是训练数据集的类别标识,你可以任意给定。对于分类，它是标识某类的整数(支持多个类)； 当然，如果是回归，这是目标值，就要实事求是了。

<index> 是以1开始的整数，可以是不连续的(某项可能为空)；

<value>就是要训练的数据，从分类的角度来说就是特征值，数据之间用空格隔开。比如: -15 1:0.708 2:1056 3:-0.3333。说明，如果特征值为0，特征冒号前面的index(姑且称做序号)可以不连续。如：-15 1:0.708 3:-0.33。表明第2个特征值为0，从编程的角度来说，这样做可以减少内存的使用，并提高做矩阵内积时的运算速度。我们平时在matlab中产生的数据都是没有序号的常规矩阵，所以为了方便最好编一个程序进行转化。

**Svmtrain和Svmpredict的用法**

LIBSVM软件提供的各种功能都是DOS命令执行方式。我们主要用到两个程序，svmtrain(训练建模)和svmpredict(使用已有的模型进行预测)，下面分别对这两个程序的使用方法、各参数的意义以及设置方法做一个简单介绍：

Svmtrain的用法：svmtrain [**options**] training\_set\_file [**model\_file**]

**Options**：可用的选项即表示的涵义如下

-s svm类型：SVM设置类型(默认0)

0 -- C-SVC

1 -- v-SVC

2 – 一类SVM

3 -- e -SVR

4 -- v-SVR

-t 核函数类型：核函数设置类型(默认2)

0 – 线性：u'v

1 – 多项式：(r\*u'v + coef0)^degree

2 – RBF函数：exp(-r|u-v|^2)

3 –sigmoid：tanh(r\*u'v + coef0)

-d degree：核函数中的degree设置(默认3)

-g r(gama)：核函数中的包含r的函数设置(默认1/ k)

-r coef0：核函数中的coef0设置(默认0)

-c cost：设置C-SVC，e-SVR和v-SVR的参数(默认1)

-n nu：设置c-SVC，一类SVM和v- SVR的参数(默认0.5)

-p e：设置e -SVR 中损失函数的值(默认0.1)

-m cachesize：设置cache内存大小，以MB为单位(默认40)

-e ?：设置允许的终止判据(默认0.001)

-h shrinking：是否使用启发式，0或1(默认1)

-wi weight：设置第几类的参数C为weight C(C-SVC中的C)(默认1)

-v n: n-fold交互检验模式

其中-g选项中的k是指输入数据中的属性数。option -v 随机地将数据剖分为n部分并计算交互检验准确度和均方根误差。以上这些参数设置可以按照SVM的类型和核函数所支持的参数进行任意组合，如果设置的参数在函数或SVM类型中没有也不会产生影响，程序不会接受该参数；如果应有的参数设置不正确，参数将采用默认值。

training\_set\_file是要进行训练的数据集(如heart\_scle)；

model\_file是训练结束后产生的模型文件 (train.model)，文件中包括支持向量样本数、支持向量样本以及lagrange系数等必须的参数；该参数如果不设置将采用默认的文件名，也可 以设置成自己惯用的文件名。

Svmpredict的用法：

**svmpredict** test\_file model\_file output\_file

model\_file是由svmtrain产生的模型文件(train.model)；test\_file是要进行预测的数据文件；Output\_file是svmpredict的输出文件。svm-predict没有其它的选项。

svmtrain -s 0 -c 1000 -t 1 -g 1 -r 1 -d 3 data\_file

训练一个由多项式核(u'v+1)^3和C=1000组成的分类器。

svmtrain -s 1 -n 0.1 -t 2 -g 0.5 -e 0.00001 data\_file

在RBF核函数exp(-0.5|u-v|^2)和终止允许限0.00001的条件下，训练一个v-SVM (u= 0.1)分类器。

svmtrain -s 3 -p 0.1 -t 0 -c 10 data\_file

以线性核函数u'v和C=10及损失函数是0.1求解SVM回归。

# LS-SVM的实现与应用

[工具包](LIB/A1_LS-SVM_MINE)(文件夹) 🡪 DEMO[多类分割程序](LIB/A1_LS-SVM_MINE/weng_multiClass.m)

clc; clear all;

% test data

TestData=importdata('test.data');

Xt\_label=TestData(:,1);

Xt\_f=TestData(:,2:end);

clear TestData;

Xt\_NumPoints=size(Xt\_label,1);

% train data

TrainData=importdata('train.data');

X\_f=TrainData(:,2:end);

X\_label=TrainData(:,1);

clear TrainData;

X\_feature\_dimension=size(X\_f,2);

X\_train=zeros(1000,X\_feature\_dimension);

y\_train=zeros(1000,1);

for i=1:10

temp=find(X\_label==(i-1));

s=size(temp,1);

% returns a vector containing a random permutation of the integers 1:N.

add\_rp=randperm(s);

add\_rp\_sub=add\_rp(1:100);

X\_train((i-1)\*100+1:i\*100,:)=X\_f(temp(add\_rp\_sub),:);

y\_train((i-1)\*100+1:i\*100)=(i-1)\*ones(100,1);

clear t1 s ra k

end

t1=cputime;

model = initlssvm(X\_train,y\_train,'c',[],[],'RBF\_kernel');

model = tunelssvm(model,'simplex','crossvalidatelssvm',{10,'misclass'},'code\_OneVsAll');

model = trainlssvm(model);

yt = simlssvm(model,Xt\_f);

t=yt-Xt\_label;

Accurancy=size(find(t==0),1)/Xt\_NumPoints;

fprintf(1,'the precision is %2.3f%%',Accurancy);

t2=cputime;

time=t2-t1;

fprintf(1,'Tuning time %i \n',time);

fprintf(1,'Accuracy: %2.2f\n',Accurancy);

# My Program Design