

Libsvm-FarutoUltimate3.0 based on libsvm-mat-2.89-3

By faruto @ faruto's Studio~

Email:patrick.lee@foxmail.com

QQ:516667408

<http://blog.sina.com.cn/faruto>

last modified 2010.01.17

Super Moderator @ www.ilovematlab.cn

1 Prologue

首先要感谢 MATLAB 中文论坛，感谢我们的 math 老大，提供这样一个学术气氛浓厚的环境，大家能在这里互相探讨问题，解决问题，并把自己的一些好的想法与大家分享。说起 MATLAB 中文论坛，我本人是在本科大三时发现的，那时运筹学期末要交大作业，我把单纯形法的一些经典算法实现了，想要用 MATLAB 做一个 GUI，进而找到了“组织”——MATLAB 中文论坛，和论坛一直走到今天。

其次要感谢 libsvm 的原始作者台湾大学的林智仁先生，没有他的 libsvm 工具箱，这些在 MATLAB 环境里面的 SVM 的辅助函数也就没有存在意义。在此给出 libsvm 的原始引用注明：

Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines, 2001. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>

也希望大家在使用我的这个 libsvm 加强工具箱时或转载我的 SVM 的辅助函数时给出原始引用注明：

```
%% 若转载请注明：  
% faruto and liyang , LIBSVM-farutoUltimateVersion  
% a toolbox with implements for support vector machines based on libsvm,2009.  
% Software available at http://www.ilovematlab.cn  
%  
% Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for  
% support vector machines, 2001. Software available at  
% http://www.csie.ntu.edu.tw/~cjlin/libsvm
```

给出原始引用注明的目的表明对原始作者的尊敬，无论是做学问还是做人，若我们用了别人的东西，最好注明，若是我们自己的原创，那就表明创新点就好。言而总之，一句话：“做人要厚道”。

说起 SVM，其实我本人的专业是应用数学，方向是模糊数学，SVM 和我专业的联系不是非常非常大，只是本科的时候在管理学院系统科学那边做过一个 EEG 相关的项目，那时候接触的 SVM，就一直学习研究至此。所以说研究 SVM 并使用 SVM 是我的个人爱好。O(∩_∩)O

2 关于 SVM 的资源汇总

SVM 相关资源汇总[matlab-libsvm-class-regress](by faruto)

<http://www.ilovematlab.cn/thread-35262-1-1.html>

注：若您对 SVM 以及 libsvm 的使用有任何疑问，我强烈建议您先到论坛里看看这个帖子，查找一下看能否找到答案，不出意外的话 95% 的问题在这里您都能找到解决办法或者相关的一些讨论。这样做利己利人，利己指的是您可以系统看到相应问题的解决办法；利人指的是若您直接在论坛里发帖提问，而此问题若已经有解决办法，就会有人重复的帮您回答，这样很耽误大家时间。

libsvm-mat-2.89-3-[farutoUltimateVersion]

<http://www.ilovematlab.cn/thread-54665-1-1.html>

注：以上链接为该加强工具箱的更新地址，您只需关注这个帖子就可以得到这个工具箱的相关更新消息了。

SVM 讲解视频汇总[by faruto]

<http://www.ilovematlab.cn/thread-62252-1-1.html>

注：这是我在论坛里面制作的相应的 SVM 教学视频，若您是新手，建议您先看看视频，这样会很快上手的。

欢迎您购买 《MATLAB 神经网络 30 个案例分析》（我是作者之一）

SVM 是该书的内容之一

购买方式：<http://www.ilovematlab.cn/thread-47939-1-1.html>

MATLAB 神经网络 30 个案例分析——目录

<http://www.ilovematlab.cn/thread-59023-1-1.html>

注：以上是论坛策划，由我们几个共同版主编写的一本书，SVM 是该书的内容之一。

3 更新说明

- (1) 使用 Sheffield 大学制作的 gatoolbox 重新编写了用 GA 来进行 SVM 参数寻优的插件函数，效果更佳。
- (2) 添加了利用启发式方法对 c, g, p（回归问题）三个参数寻优的函数。
- (3) 将所有的插件函数进行了重新整合，封装了统一的函数接口，SVC.m 和 SVR.m 来处理分类问题和回归问题。
- (4) 制作了详细的使用说明介绍 TutorialForFarutoUltimate3.0.pdf。

注：从此次工具箱更新开始将不提供原始的 m 文件源代码，而只提供 pcode，若您想要源代码，有以下几种方式：

- (1) 若您是《MATLAB 神经网络 30 个案例分析》书籍的购买者，我将免费赠送您源代码。您只需留下邮箱，我核实若您书籍购买者，我会将源代码发到您的邮箱里面

关于该书

购买方式：<http://www.ilovematlab.cn/thread-47939-1-1.html>

MATLAB 神经网络 30 个案例分析——目录

<http://www.ilovematlab.cn/thread-59023-1-1.html>

- (2) 单独购买源代码

价钱：20RMB （看到自己的代码才 20 元出售，挺杯具的，但没办法总得定一个价位）

【人是社会中的人，我也得过活，我并没有那么高尚，人总得现实点。】

通知支付宝支付：我的支付宝账号 farutoliyang@gmail.com

您支付后，只需留下你的支付宝账号并将交易截图贴出来，并留下邮箱，我看到核实后会将源代码发到您的邮箱里面。

- (3) 和我熟识的人，若有需要，我也会免费赠送源代码。O(∩_∩)O

其实若您只是使用的话，其实无需源代码。若想要源代码，我建议您通过第一种方式，这个是最划算的。（其实后面两种只是噱头，俺真正的目的是想让您买本书，那是一本超赞的书~）

4 辅助函数插件内容列表

a_template_flow_usingSVM_class.m

脚本代码，一个利用 SVM 来分类的过程模板。

a_template_flow_usingSVM_regress.m

脚本代码，一个利用 SVM 来回归的过程模板。

fasticaForSVM.m

ica 降维预处理函数

gaSVMcgForClass.m

对于分类问题利用 GA 来进行参数优化 (c, g)。

gaSVMcgForRegress.m

对于回归问题利用 GA 来进行参数优化 (c, g)。

gaSVMcgpForRegress.m

对于回归问题利用 GA 来进行参数优化 (c, g, p)。

pcaForSVM.m

pca 降维预处理函数

psoSVMcgForClass.m

对于分类问题利用 PSO 来进行参数优化 (c, g)。

psoSVMcgForRegress.m

对于回归问题利用 PSO 来进行参数优化 (c, g)。

scaleForSVM.m

归一化预处理函数

SVMcgForClass.m

对于分类问题网格参数优化 (c, g)。

SVMcgForRegress.m

对于回归问题网格参数优化 (c, g)。

svmplot.m

分类问题的可视化图 (分类超平面绘制仅对二维问题分类标签为-1 和+1 的可用)。

VF.m

对于分类问题的一些评价指标 (如准确率, 正类准确率, 负类准确率 bla,bla ...)。

SVC.m

对于分类问题各种函数插件的一个整合接口

SVR.m

对于回归问题各种函数插件的一个整合接口

TutorialTest.m

使用说明介绍文件 TutorialForFarutoUltimate3.0.pdf 中的各种测试代码段。

5 辅助函数接口介绍以及测试

[train_scale,test_scale,ps] = scaleForSVM(train_data,test_data,ymin,ymax)

输入:

train_data: 训练集
test_data: 测试集
ymin: 归一化范围下限 (可不输入, 默认为 0)
ymax: 归一化范围上限 (可不输入, 默认为 1)

输出:
train_scale: 归一化后的训练集
test_scale: 归一化后的测试集
ps: 归一化映射

测试代码:
train_data = [1 12;3 4;7 8]
test_data = [9 10;6 2]

[train_scale,test_scale,ps] = scaleForSVM(train_data,test_data,0,1)

运行结果:
train_data =
 1 12
 3 4
 7 8
test_data =
 9 10
 6 2
train_scale =
 0 1.0000
 0.2500 0.2000
 0.7500 0.6000
test_scale =
 1.0000 0.8000
 0.6250 0
ps =
 name: 'mapminmax'
 xrows: 2
 xmax: [2x1 double]
 xmin: [2x1 double]
 xrange: [2x1 double]
 yrows: 2
 ymax: 1
 ymin: 0
yrange: 1

说明: 归一化并不是必须采用的预处理方法。但一旦采用了, 这个步骤就十分重要, 因为这是使用 **SVM** 的第一步骤, 原始数据从这里将会被变化, 若处理不当会使后面的分类或回归效果不佳。

原始数据到底该怎么进行归一化，我想到的是以下几个问题：

- (1) 是对每一个样本进行进行归一化（按行归一化）还是对每一个维度进行归一化（按列归一化）？
- (2) 是将训练集和测试集分别归一化还是放在一起形成一个大矩阵一起进行归一化？

对于上面的我个人的理解和给出的解决办法是：

- (1) 对每一个维度进行归一化

理由：对于每个样本，由于它的每一个维度的量纲不同，若对每一个样本进行归一化且在量纲数量级差别悬殊时会使样本中较低数量级的属性变为 0，会使原始信息过多丧失。比如：

```
sample =
    1     2     0.5    100000
    5     7     0.4    200000
```

若按行对每一个样本进行[0, 1]归一化（按行归一化），则结果为：

```
sample_scale =
    0.0000    0.0000         0    1.0000
    0.0000    0.0000         0    1.0000
```

你会看到由于数量级的差别，对于每一样本的前三维的数据都被归一化为 0，这样其实是不合理，会使原始数据的信息过多丢失。但若采用对每一维度进行归一化，就不会大范围发生这种情况，因为对于同一维度，量纲级别是相同的。对每一维度进行[0, 1]归一化（按列归一化），结果为：

```
sample_scale =
    0     0     1     0
    1     1     0     1
```

- (2) 将训练集和测试集放在一起，一起进行归一化。

理由：用测试代码中的例子

```
train_data =
    1    12
    3     4
    7     8

test_data =
    9    10
    6     2
```

若先将训练集进行归一化（按每一维度进行），然后把这个归一化映射记录下来，当有测试集时再用这个归一化映射对测试集进行归一化，对训练集进行归一化时对于第一维度归一化映射记录的最大值是 7，这就接受一个假设是所有数据的第一维度的最大值不能超过 7，但我们看到新的测试集拿来的时候它的第一维度的值不一定非得小于 7，测试数据中的测试集的第一维度的最大值为 9>7。即若分别归一化会产生这种不合理的现象。将训练集和测试集放在一起归一化就可以避免这种情况，统一归一化时每一维度的最大值和最小值是从训练集和测试集中寻找。

关于这个问题，我和论坛的一些朋友曾一起详细讨论过，**有的朋友说这样做不合理**，因为这样做的话测试集间接参与的 SVM 模型的建立，他们的意思

是 SVM 的整体思路是用训练集建立 SVM 模型,然后用这个模型来预测测试集;或者说得更明白一点就是用训练集建立一个固定的 SVM 模型,然后每来一个测试集,就可以用这个固定 SVM 模型来进行预测。**对此,我个人的给出的解释和理解是**,我们处理的问题是面向数据的,当有新的测试集来的时候,**若我们能建立一个更加适合这个测试集的 SVM 模型来预测**,这样岂不是更好,即将原始训练集和新来的测试集放在一起统一归一化,这样得到的 SVM 模型更加适合当前的测试集,当又有新的测试集来的时候,我们再将这个新的测试集和原始的训练集放在重新归一化,再进行后续的 SVM 步骤,建立 SVM 模型。

以上归一化问题是见仁见智的,您可以保留自己的意见,以上是我个人的见解,事实证明将训练集和测试集放在一起归一化的效果绝对要比分别归一化的效果要好很多。您可以自行验证测试,用事实说话。

[train_pca,test_pca] = pcaForSVM(train,test,threshold)

输入:

train_data: 训练集,格式要求与 svmtrain 相同。

test_data: 测试集,格式要求与 svmtrain 相同。

threshold: 对原始变量的解释程度 ([0, 100]之间的一个数),通过该阈值可以选取出主成分,该参数可以不输入,默认为 90,即选取的主成分默认可以达到对原始变量达到 90%的解释程度。

输出:

train_pca: 进行 pca 降维预处理后的训练集。

test_pca: 进行 pca 降维预处理后的测试集。

测试代码:

```
load wine_test
```

```
whos train_data
```

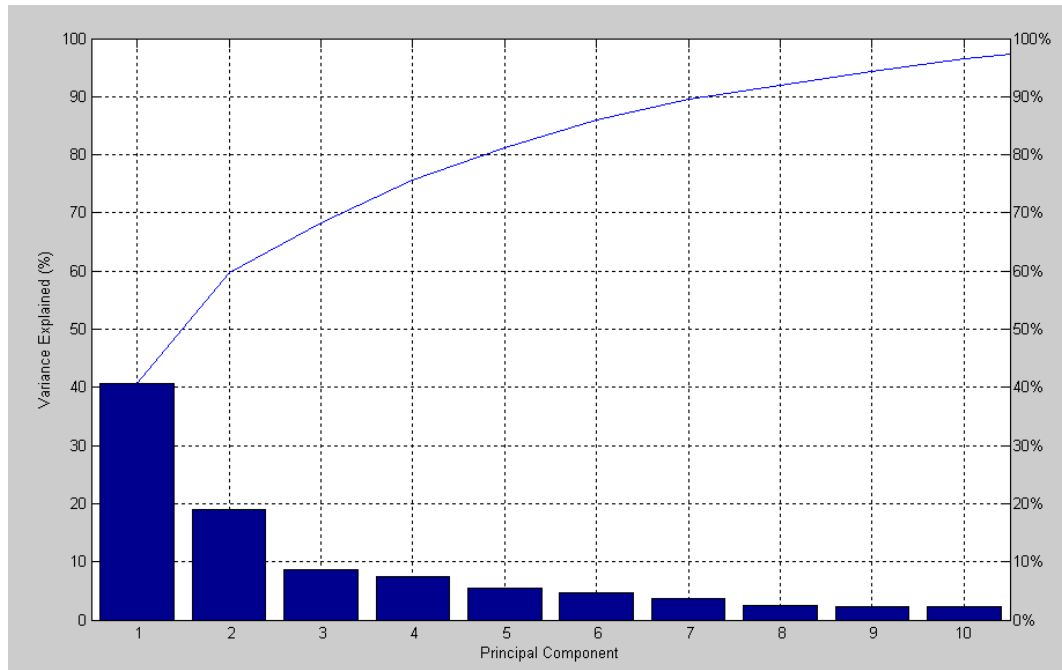
```
[train_scale,test_scale,ps] = scaleForSVM(train_data,test_data,0,1);
```

```
[train_pca,test_pca] = pcaForSVM(train_scale,test_scale,95);
```

```
whos train_pca
```

运行结果:

Name	Size	Bytes	Class	Attributes
train_data	89x13	9256	double	
train_pca	89x10	7120	double	



**[bestacc,bestc,bestg]=SVMcgForClass(train_label,train,cmin,cmax,gmin,gmax,v,
cstep,gstep,accstep)**

输入:

train_label: 训练集的标签, 格式要求与 `svmtrain` 相同。

train: 训练集, 格式要求与 `svmtrain` 相同。

cmin, cmax: 惩罚参数 c 的变化范围, 即在 $[2^{cmin}, 2^{cmax}]$ 范围内寻找最佳的参数 c , 默认值为 $cmin=-8$, $cmax=8$, 即默认惩罚参数 c 的范围是 $[2^{(-8)}, 2^8]$ 。

gmin, gmax: RBF 核参数 g 的变化范围, 即在 $[2^{gmin}, 2^{gmax}]$ 范围内寻找最佳的 RBF 核参数 g , 默认值为 $gmin=-8$, $gmax=8$, 即默认 RBF 核参数 g 的范围是 $[2^{(-8)}, 2^8]$ 。

v: 进行 Cross Validation 过程中的参数, 即对训练集进行 v -fold Cross Validation, 默认为 3, 即默认进行 3 折 CV 过程。

cstep, gstep: 进行参数寻优是 c 和 g 的步进大小, 即 c 的取值为 2^{cmin} , $2^{(cmin+cstep)}$, ..., 2^{cmax} , g 的取值为 2^{gmin} , $2^{(gmin+gstep)}$, ..., 2^{gmax} , 默认取值为 $cstep=1$, $gstep=1$ 。

accstep: 最后参数选择结果图中准确率离散化显示的步进间隔大小 ($[0, 100]$ 之间的一个数), 默认为 4.5。

输出:

bestCVaccuracy: 最终 CV 意义下的最佳分类准确率。

bestc: 最佳的参数 c 。

bestg: 最佳的参数 g 。

测试代码:


```
load wine_test
```

```
[train_scale,test_scale,ps] = scaleForSVM(train_data,test_data,0,1);
```

```
[bestacc,bestc,bestg]=SVMcgForClass(train_data_labels,train_scale,-10,10,-10,10,5,0  
.5,0.5,4.5)
```

运行结果:

```
bestacc =
```

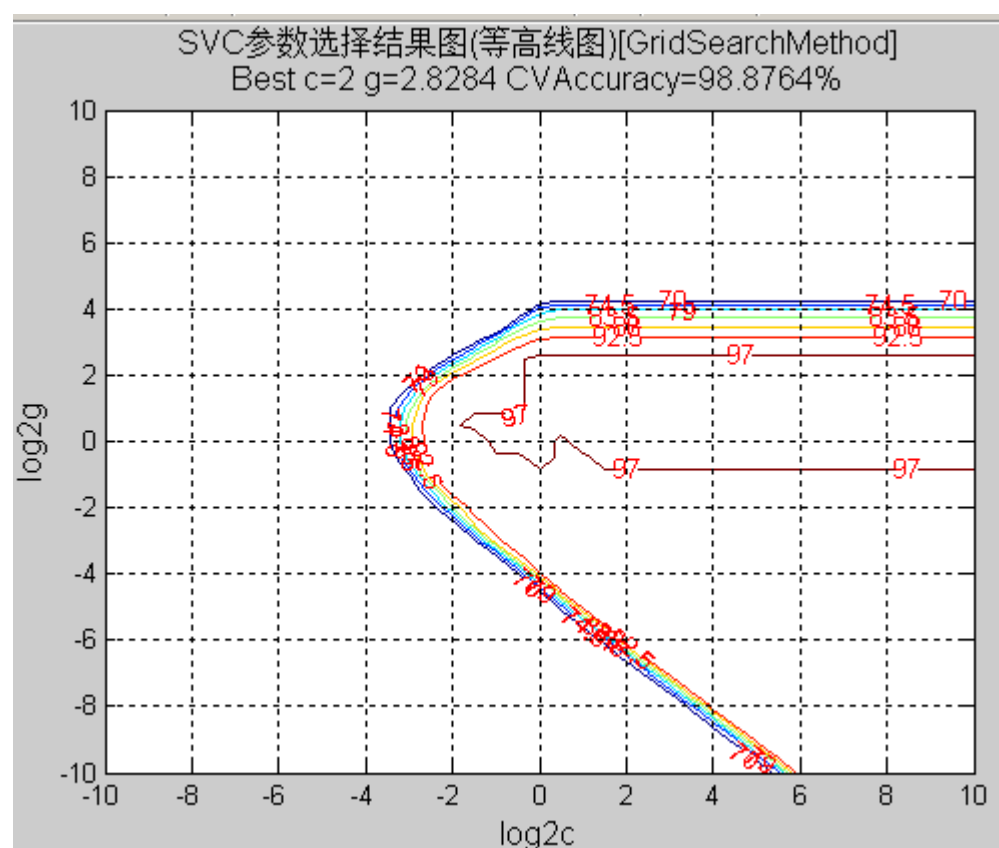
```
98.8764
```

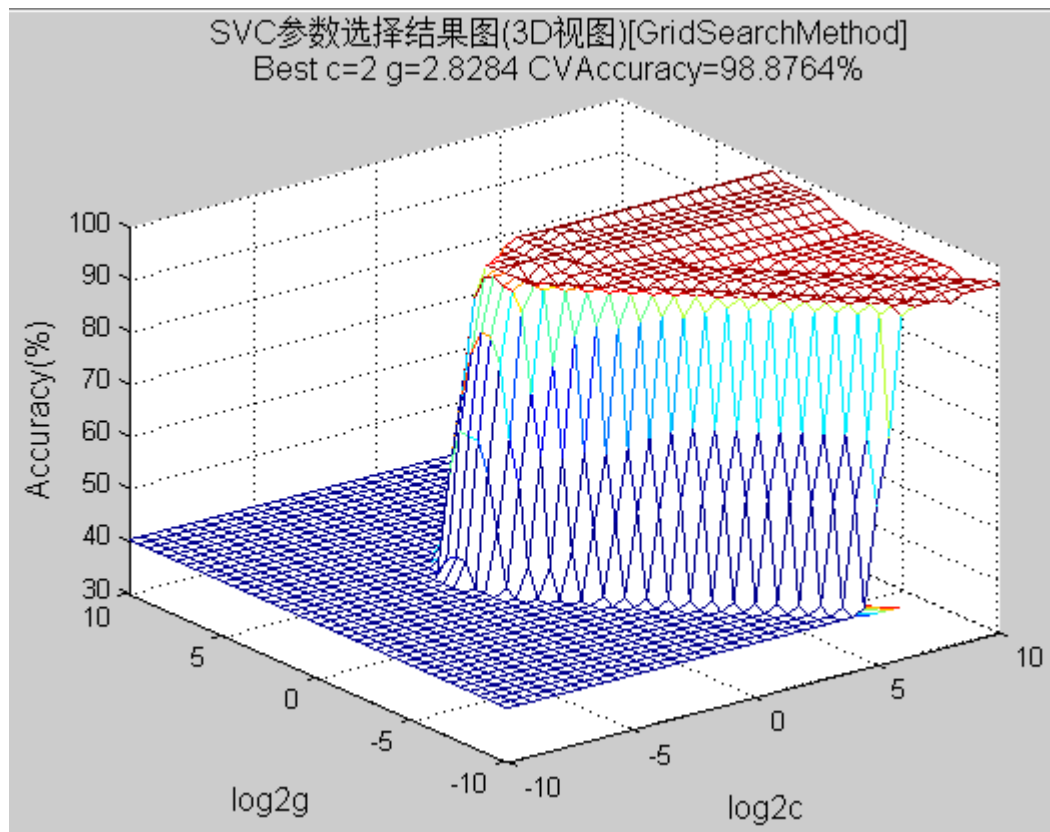
```
bestc =
```

```
2
```

```
bestg =
```

```
2.8284
```





[bestCVmse, bestc, bestg]= SVMcgForRegress(train_label, train,

cmin, cmax, gmin, gmax, v, cstep, gstep, msestep)

其输入输出与 SVMcgForClass 类似，这里不再赘述。

[bestCVaccuracy, bestc, bestg, pso_option]= psoSVMcgForClass(train_label,

train, pso_option)

输入：

train_label: 训练集的标签，格式要求与 svmtrain 相同。

train: 训练集，格式要求与 svmtrain 相同。

pso_option: PSO 中的一些参数设置，可不输入，有默认值，详细请看代码的帮助说明。

输出：

bestCVaccuracy: 最终 CV 意义下的最佳分类准确率。

bestc: 最佳的参数 c。

bestg: 最佳的参数 g。

pso_option: 记录 PSO 中的一些参数。

测试代码:

load [wine_test](#)

```
[train_scale,test_scale,ps] = scaleForSVM(train_data,test_data,0,1);
```

```
pso_option.c1 = 1.5;  
pso_option.c2 = 1.7;  
pso_option.maxgen = 100;  
pso_option.sizepop = 20;  
pso_option.k = 0.6;  
pso_option.wV = 1;  
pso_option.wP = 1;  
pso_option.v = 3;  
pso_option.popcmax = 100;  
pso_option.popcmin = 0.1;  
pso_option.popgmax = 100;  
pso_option.popgmin = 0.1;
```

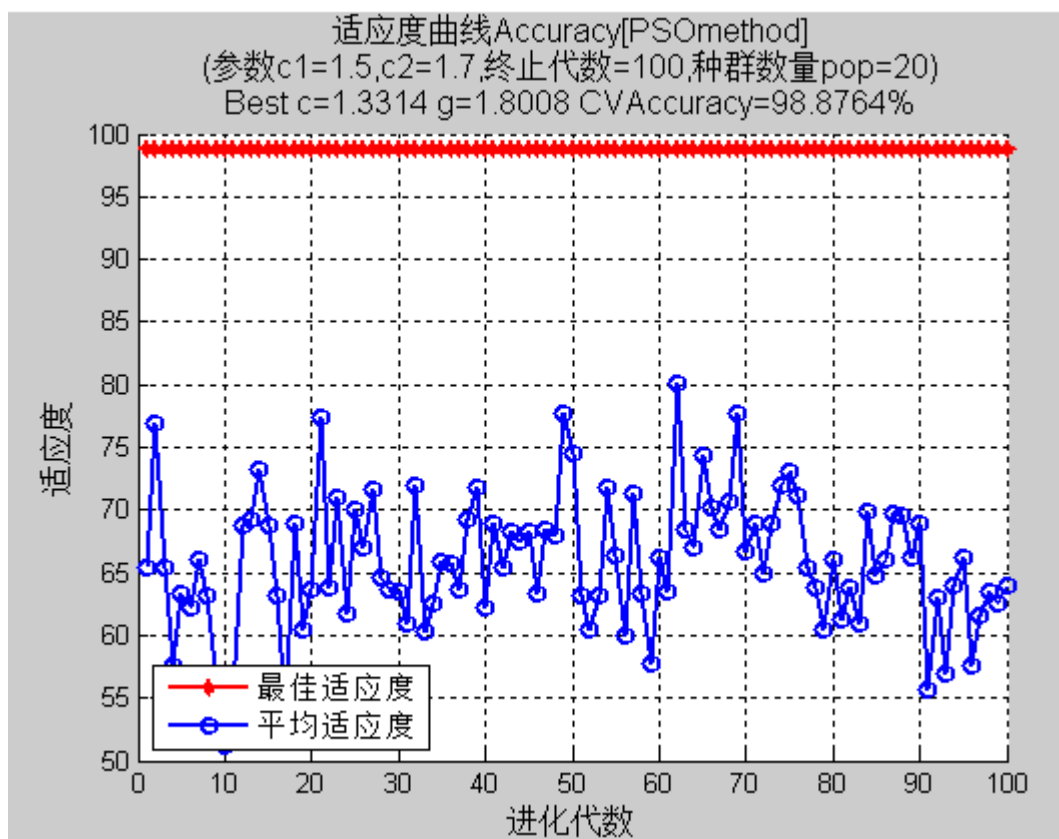
```
[bestacc,bestc,bestg]=psoSVMcgForClass(train_data_labels,train_scale,pso_option)
```

运行结果:

```
bestacc =  
    98.8764
```

```
bestc =  
    1.3314
```

```
bestg =  
    1.8008
```



```
[bestCVmse, bestc, bestg, pso_option]= psoSVMcgForRegress(train_label, train,
pso_option)
```

其输入输出与 psoSVMcgForClass 类似，这里不再赘述。

```
[bestCVaccuracy, bestc, bestg, ga_option]= gaSVMcgForClass(train_label,
train, ga_option)
```

输入：

train_label: 训练集的标签，格式要求与 svmtrain 相同。

train: 训练集，格式要求与 svmtrain 相同。

ga_option: GA 中的一些参数设置，可不输入，有默认值，详细请看代码的帮助说明。

输出：

bestCVaccuracy: 最终 CV 意义下的最佳分类准确率。

bestc: 最佳的参数 c。

bestg: 最佳的参数 g。

ga_option: 记录 GA 中的一些参数。

测试代码:

load wine_test

```
[train_scale,test_scale,ps] = scaleForSVM(train_data,test_data,0,1);
```

```
ga_option.maxgen = 100;
```

```
ga_option.sizepop = 20;
```

```
ga_option.cbound = [0,100];
```

```
ga_option.gbound = [0,100];
```

```
ga_option.v = 10;
```

```
ga_option.ggap = 0.9;
```

```
[bestacc,bestc,bestg]=gaSVMcgForClass(train_data_labels,train_scale,ga_option)
```

运行结果:

bestacc =

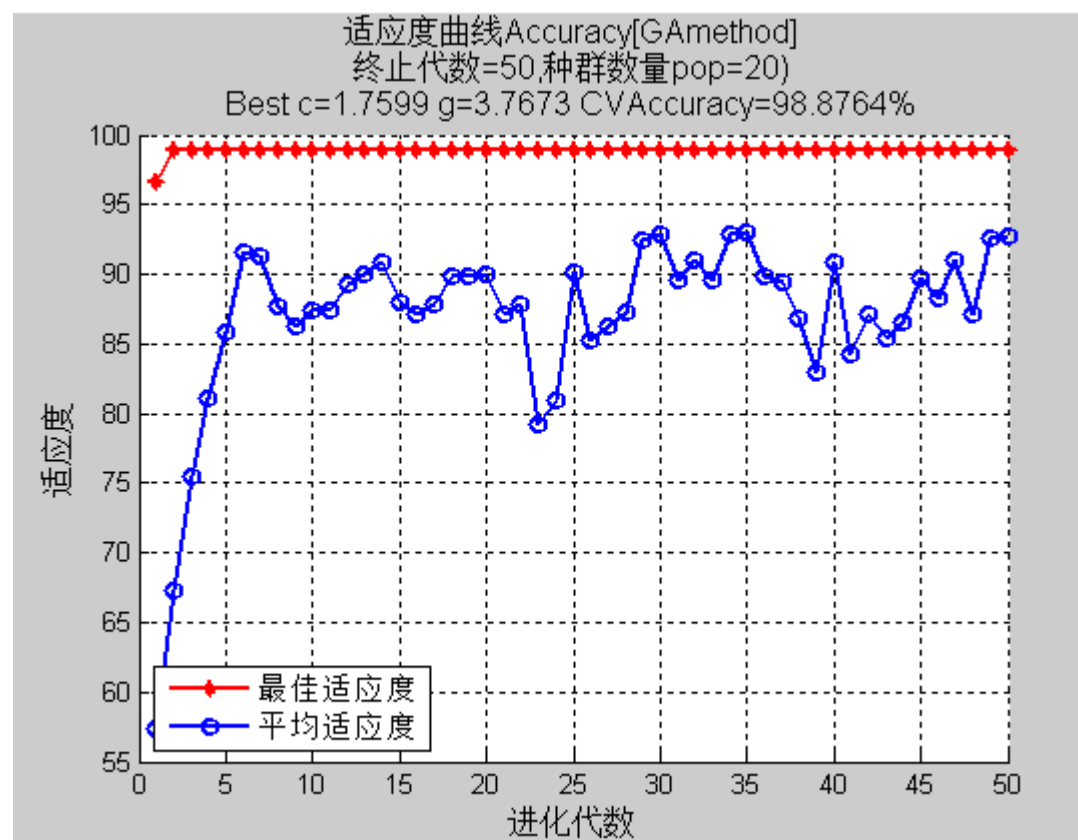
98.8764

bestc =

1.7599

bestg =

3.7673

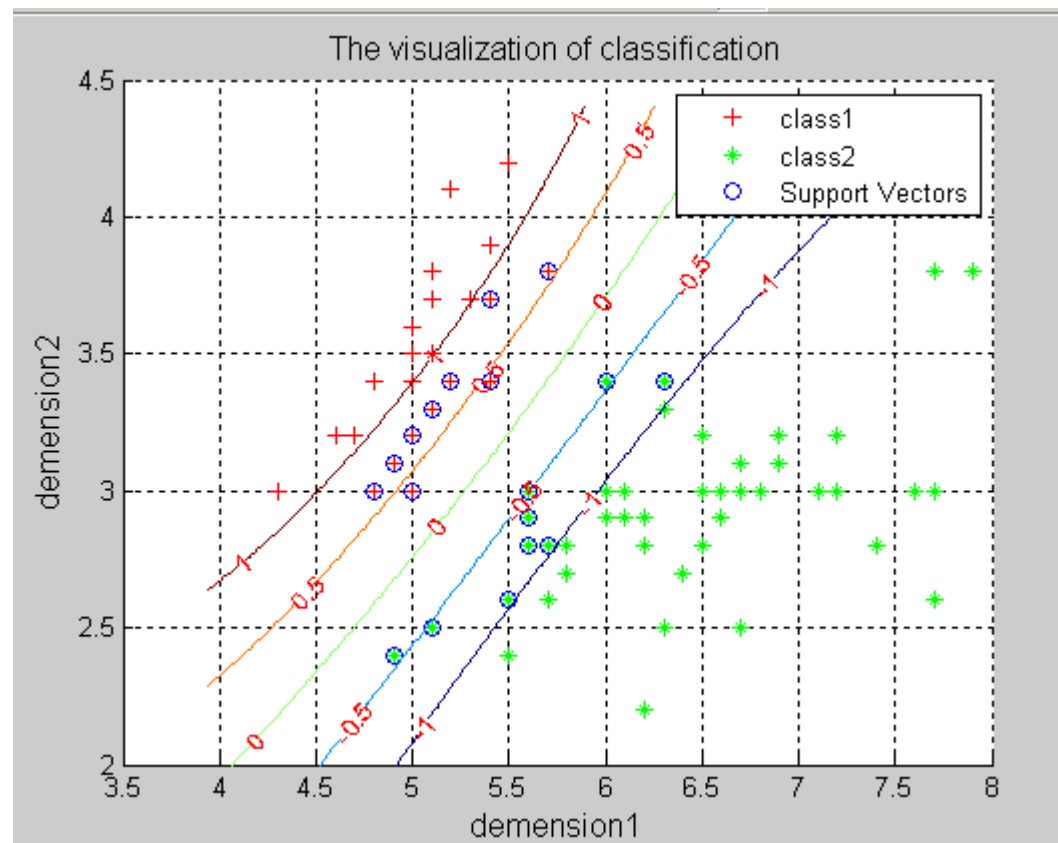


svmplot(labels,dataset,model)

测试代码：

```
load fisheriris;
data = [meas(:,1), meas(:,2)];
groups = ismember(species,'setosa');
[train, test] = crossvalind('holdOut',groups);
dataset = data(train,:);
labels = double(groups(train));
model = svmtrain(labels,dataset,'-c 2 -g 0.1');
figure;
grid on;
svmplot(labels,dataset,model);
```

运行结果：



```
[predict_label,accuracy]=SVC(train_label,train_data,test_label,test_data,Method_option)
```

```
% =====input=====
1 表示 是 0表示 否
% Method_option.plotOriginal = 0 or 1 表示是否画出原始数据
% Method_option.scale = 0 or 1 表示是否进行归一化
% Method_option.plotScale = 0 or 1 表示是否画出归一化后的图像
% Method_option.pca = 0 or 1 表示是否进行pca降维预处理
% Method_option.type = 1[grid] or 2[ga] or 3[psa] 表示采用何种寻参方法
```

测试代码:

```
load wine_test;
train_label = train_data_labels;
train_data = train_data;
test_label = test_data_labels;
test_data = test_data;
```

```
Method_option.plotOriginal = 0;
Method_option.scale = 1;
Method_option.plotScale = 0;
Method_option.pca = 1;
Method_option.type = 1;
```

```
[predict_label,accuracy] =
SVC(train_label,train_data,test_label,test_data,Method_option);
```

运行结果:

```
bestCVaccuracy =
    97.7528
bestc =
    1.7411
bestg =
    5.2780
Accuracy = 100% (89/89) (classification)
Accuracy = 97.7528% (87/89) (classification)
accuracy =
    100.0000    97.7528
```

```
[predict_Y,mse,r] = SVR(train_y,train_x,test_y,test_x,Method_option)
```

```
% =====input=====
1 表示 是 0表示 否
% Method_option.plotOriginal = 0 or 1 是否画出原始数据
% Method_option.xscale = 0 or 1 是否对自变量归一化
% Method_option.yscale = 0 or 1 是否对因变量归一化
% Method_option.plotScale = 0 or 1 是否画出归一化后的因变量
% Method_option.pca = 0 or 1 是否进行pca降维预处理
% Method_option.type = 1[grid cg] or 2[ga cg] or 3[pso cg] or 4[pso cgp] or 5[ga
cgp]
表示采用何种寻参方法
```

测试代码:

```
load x123;
train_y = x1(1:17);
train_x = [1:17]';
test_y = x1(18:end,:);
test_x = [18:20]';
```

```
Method_option.plotOriginal = 0;
Method_option.xscale = 1;
Method_option.yscale = 1;
Method_option.plotScale = 0;
Method_option.pca = 0;
Method_option.type = 5;
```

```
[predict_Y,mse,r] = SVR(train_y,train_x,test_y,test_x,Method_option);
```

运行结果:

```
bestCVmse =
```

```
0.0234
```

```
bestc =
```

```
58.8300
```

```
bestg =
```

```
47.2991
```

```
bestp =
```

```
0.0989
```

```
Mean squared error = 0.00700163 (regression)
```

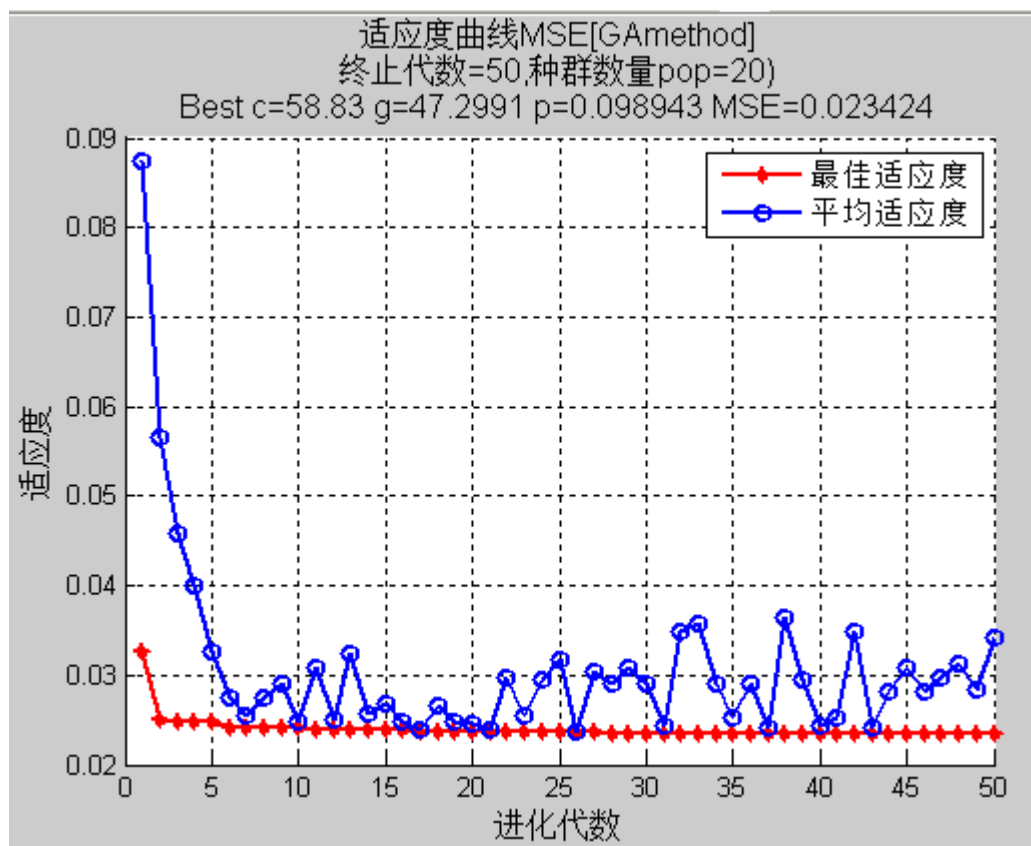
```
Squared correlation coefficient = 0.946016 (regression)
```

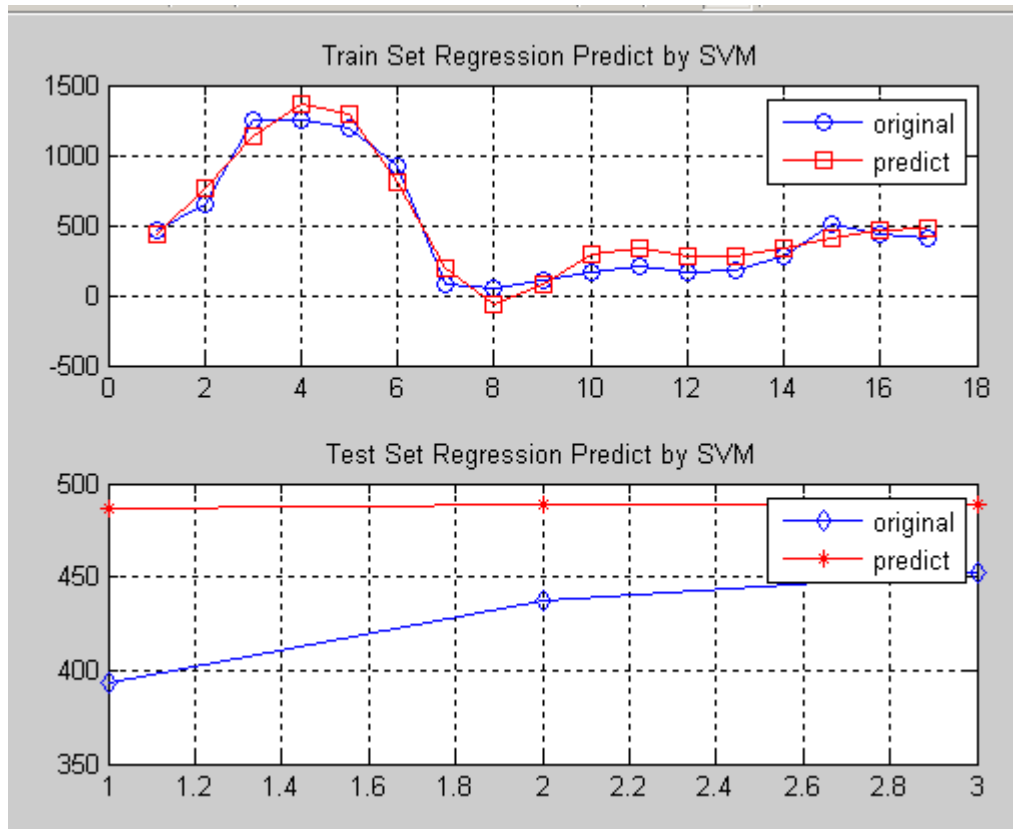
```
Mean squared error = 0.00292977 (regression)
```

```
Squared correlation coefficient = 0.991936 (regression)
```

```
mse =
```


0.0070 0.0029
r =
0.9460 0.9919





```
[score,str] = VF(true_labels,predict_labels,type)
```

```
% VF : validation_function
% Note:This tool is designed only for binary-class C-SVM with labels
% {1,-1}. Multi-class,regression and probability estimation are not supported.
% by faruto 2009.12.03
% Email:patrick.lee@foxmail.com QQ:516667408
% type : 1,2,3,4,5
% 1:Accuracy = #true / #total
% 2:Precision = true_positive / (true_positive + false_positive)
% 3:Recall = true_positive / (true_positive + false_negative)
% 4:F-score = 2 * Precision * Recall / (Precision + Recall)
% 5:
% BAC (Ballanced ACcuracy) = (Sensitivity + Specificity) / 2,
% where Sensitivity = true_positive / (true_positive + false_negative)
% and Specificity = true_negative / (true_negative + false_positive)
```

测试代码:

```
load wine_test
[train_scale,test_scale,ps] = scaleForSVM(train_data,test_data,0,1);
model = svmtrain(train_data_labels,train_scale,'-c 2 -g 0.1');
```

```
[pre,acc] = svmpredict(train_data_labels,train_scale,model);
[score,str] = VF(train_data_labels,pre,1)
[score,str] = VF(train_data_labels,pre,2)
[score,str] = VF(train_data_labels,pre,3)
[score,str] = VF(train_data_labels,pre,4)
[score,str] = VF(train_data_labels,pre,5)
```

运行结果:

score =

96.6292

str =

Accuracy = 96.6292% (86/89) [Accuracy = #true / #total]

score =

96.7742

str =

Precision = 96.7742% (30/31) [Precision = true_positive / (true_positive + false_positive)]

score =

100

str =

Recall = 100% (30/30) [Recall = true_positive / (true_positive + false_negative)]

score =

98.3607

str =

F-score = 98.3607% [F-score = 2 * Precision * Recall / (Precision + Recall)]

score =

50

str =

BAC = 50% [BAC (Ballanced ACcuracy) = (Sensitivity + Specificity) / 2]