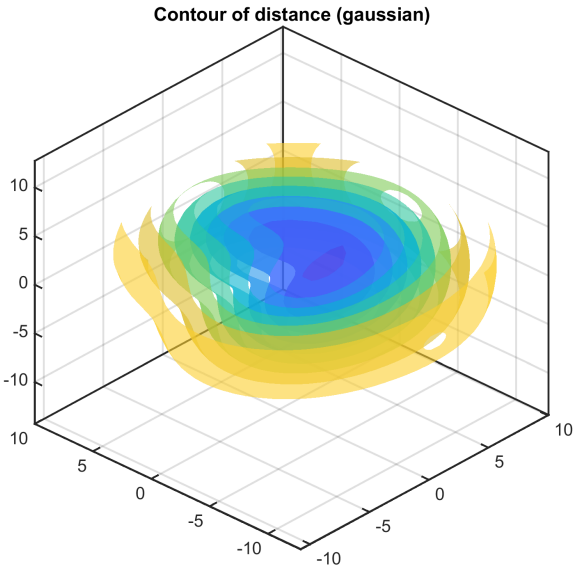
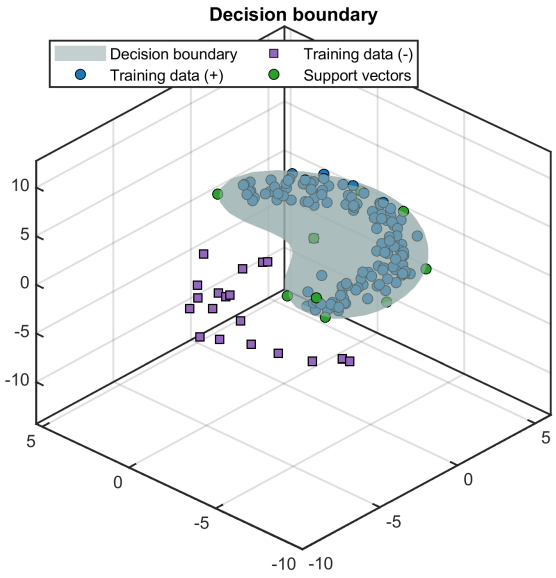




Support Vector Data Description (SVDD)

利用 SVDD 实现数据的异常检测

Version 2.1, 11-MAY-2021

Email: iqiukp@outlook.com

主要特点

- 支持单值分类和二值分类的超球体构建
- 支持多种核函数（linear, gaussian, polynomial, sigmoid, laplacian）
- 支持 2D 或 3D 数据的决策边界可视化
- 支持基于贝叶斯超参数优化、遗传算法和粒子群算法的 SVDD 的参数优化
- 支持加权的 SVDD

注意

- SVDD V2.1 仅支持 **R2016b** 以上的 MATLAB 版本
- 正样本和负样本对应的标签分别为 1 和 -1
- 提供了多个示例文件，每个文件的开头都有对应的介绍
- 此代码仅供参考

使用说明

01. 香蕉型数据集

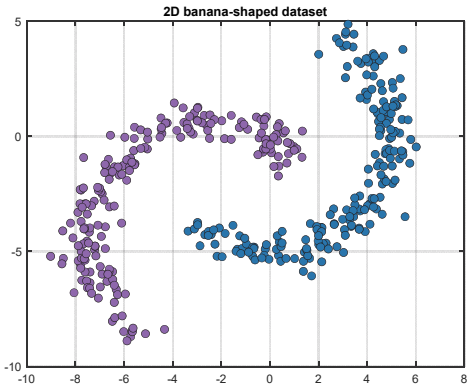
类 ***DataSet*** 用于生成和划分香蕉型数据集，其中：

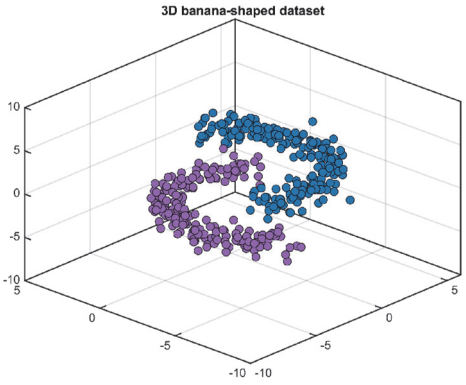
'dim' 的值代表香蕉型数据集的维度（2 或 3）；'type' 的值代表训练集的类型；

'single' 代表训练集只有正样本，'hybrid' 代表训练集同时包含正样本和负样本。

```
[data, label] = DataSet.generate;
[data, label] = DataSet.generate('dim', 2);
[data, label] = DataSet.generate('dim', 2, 'num', [200, 200]);
```

```
[data, label] = DataSet.generate('dim', 3, 'num', [200, 200], 'display'
, 'on');

% 'single' --- The training set contains only positive samples.
[trainData, trainLabel, testData, testLabel] = DataSet.partition(data,
label, 'type', 'single');

% 'hybrid' --
- The training set contains positive and negetive samples.
[trainData, trainLabel, testData, testLabel] = DataSet.partition(data,
label, 'type', 'hybrid');
```





02. 核函数

类 ***Kernel*** 用于计算核函数矩阵:

```
%{
        type    -

        linear      :  k(x,y) = x'*y
        polynomial  :  k(x,y) = (γ*x'*y+c)^d
        gaussian    :  k(x,y) = exp(-γ*||x-y||^2)
        sigmoid     :  k(x,y) = tanh(γ*x'*y+c)
        laplacian   :  k(x,y) = exp(-γ*||x-y||)


        degree -  d
        offset -  c
        gamma  -  γ
%}
kernel = Kernel('type', 'gaussian', 'gamma', value);
kernel = Kernel('type', 'polynomial', 'degree', value);
kernel = Kernel('type', 'linear');
```

```
kernel = Kernel('type', 'sigmoid', 'gamma', value);
kernel = Kernel('type', 'laplacian', 'gamma', value);
```

例如，计算 **X** 和 **Y** 的高斯核函数矩阵：

```
X = rand(5, 2);
Y = rand(3, 2);
kernel = Kernel('type', 'gaussian', 'gamma', 2);
kernelMatrix = kernel.computeMatrix(X, Y);
>> kernelMatrix

kernelMatrix =

    0.5684    0.5607    0.4007
    0.4651    0.8383    0.5091
    0.8392    0.7116    0.9834
    0.4731    0.8816    0.8052
    0.5034    0.9807    0.7274
```

03-1. 仅包含正样本的 SVDD 模型

```
[data, label] = DataSet.generate('dim', 3, 'num', [200, 200], 'display'
, 'on');
[trainData, trainLabel, testData, testLabel] = DataSet.partition(data,
label, 'type', 'single');
kernel = Kernel('type', 'gaussian', 'gamma', 0.2);
cost = 0.3;
svddParameter = struct('cost', cost,...
                       'kernelFunc', kernel);
svdd = BaseSVDD(svddParameter);

% train SVDD model
svdd.train(trainData, trainLabel);
% test SVDD model
results = svdd.test(testData, testLabel);
```

svdd.train 的输入参数只有训练数据时，会将对应的标签设置为 1.

SVDD 模型的训练和测试结果为：

```
*** SVDD model training finished ***
running time            = 0.0069 seconds
iterations              = 9
number of samples       = 140
number of SVs           = 23
```

```
ratio of SVs            = 16.4286%
accuracy                = 95.0000%


*** SVDD model test finished ***
running time            = 0.0013 seconds
number of samples       = 260
number of alarm points  = 215
accuracy                = 94.2308%
```

03-2. 包含正样本和负样本的 SVDD 模型

```
[data, label] = DataSet.generate('dim', 3, 'num', [200, 200], 'display'
, 'on');
[trainData, trainLabel, testData, testLabel] = DataSet.partition(data,
label, 'type', 'hybrid');
kernel = Kernel('type', 'gaussian', 'gamma', 0.05);
cost = 0.9;
svddParameter = struct('cost', cost,...
                       'kernelFunc', kernel);
svdd = BaseSVDD(svddParameter);

% train SVDD model
svdd.train(trainData, trainLabel);
% test SVDD model
results = svdd.test(testData, testLabel);
```

SVDD 模型的训练和测试的结果为：

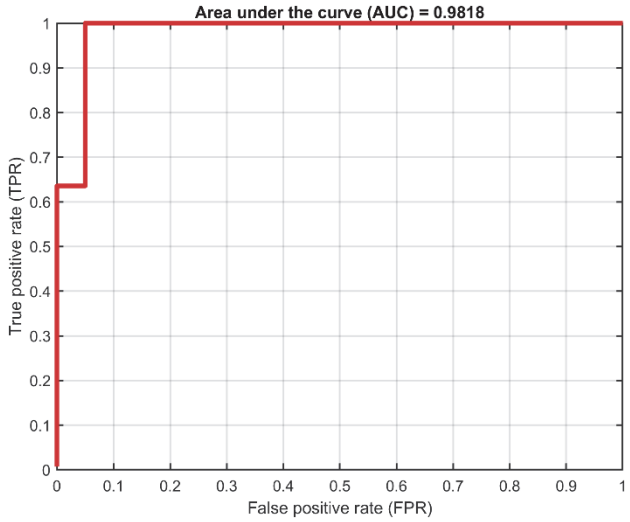
```
*** SVDD model training finished ***
running time            = 0.0074 seconds
iterations              = 9
number of samples       = 160
number of SVs           = 12
ratio of SVs            = 7.5000%
accuracy                = 97.5000%


*** SVDD model test finished ***
running time            = 0.0013 seconds
number of samples       = 240
number of alarm points  = 188
accuracy                = 96.6667%
```

04. 可视化

类 ***SvddVisualization*** 定义了训练结果和测试结果的可视化方法。

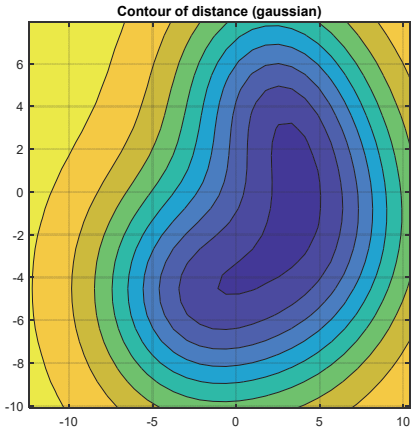
例如，训练数据的 ROC 曲线为（仅适用于训练集中包含正样本和负样本的 SVDD 模型）：

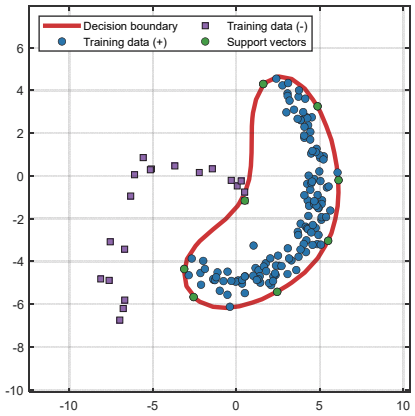
```
% Visualization
svplot = SvddVisualization();
svplot.ROC(svdd);
```

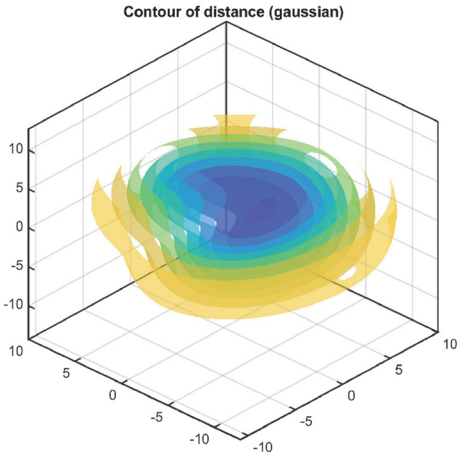


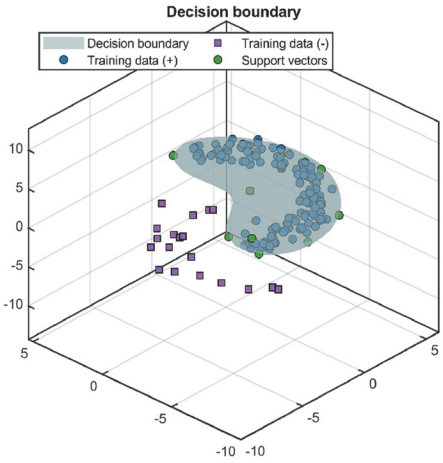
训练数据的决策边界为（仅适用于 2D 或 3D 数据）：

```
% Visualization
svplot = SvddVisualization();
svplot.boundary(svdd);
```



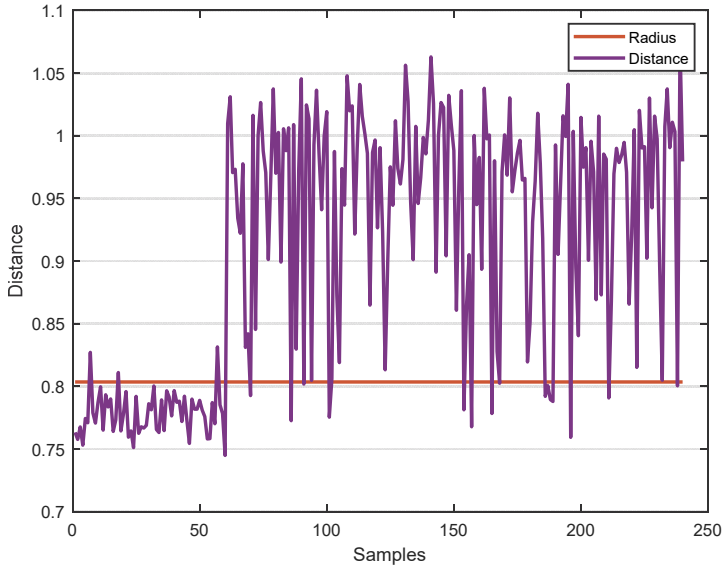






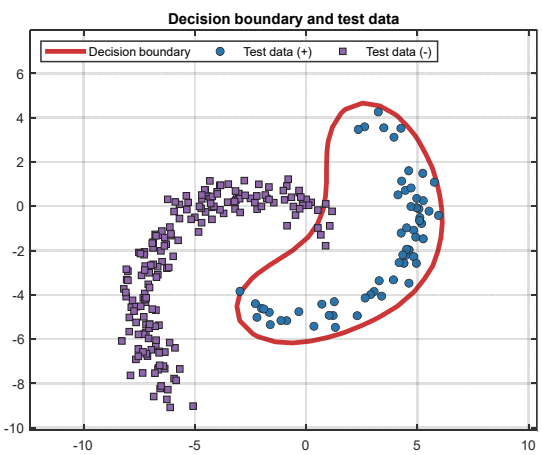
测试数据的球心距为：

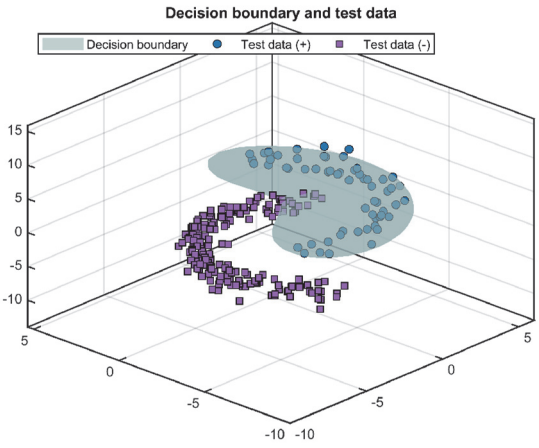
```
svplot.distance(svdd, results);
```



测试数据与决策边界（仅适用于 2D 或 3D 的数据）：

```
svplot.testDataWithBoundary(svdd, results);
```





05. 参数优化

类 ***SvddOptimization*** 用于优化 SVDD 的参数，优化算法通过结构体 *optimization* 来描述，结构体的属性包括:

优化算法：*method*

优化参数的名字：*variableName*

优化参数的类型：*variableType*

优化参数的下限：*lowerBound*

优化参数的上限：*upperBound*

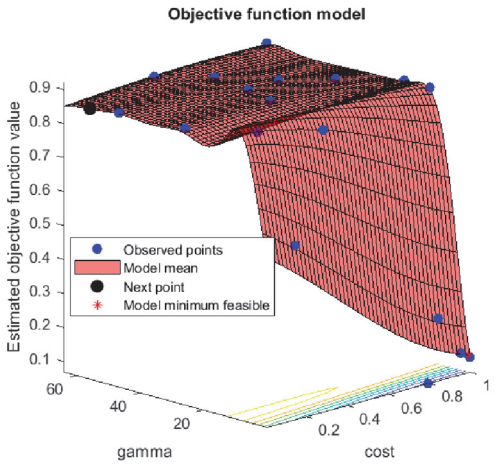
优化算法的迭代次数上限：*maxIteration*

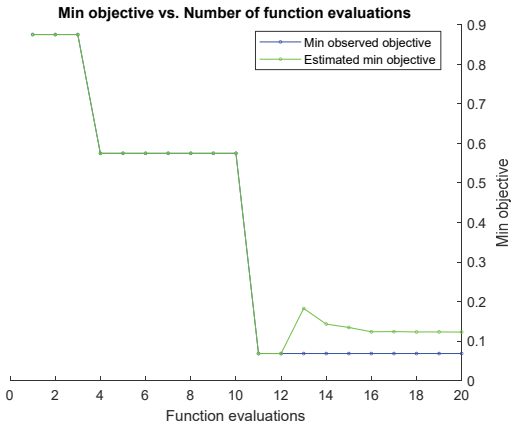
优化算法每次迭代的种子数：*points*

优化参数过程可视化：*display*

```
% optimization setting
optimization.method = 'bayes'; % bayes, ga  pso
optimization.variableName = { 'cost', 'gamma'};
optimization.variableType = {'real', 'real'}; % 'integer' 'real'
optimization.lowerBound = [10^-2, 2^-6];
optimization.upperBound = [10^0, 2^6];
optimization.maxIteration = 20;
optimization.points = 10;
optimization.display = 'on';

% SVDD parameter
svddParameter = struct('cost', cost,...
                       'kernelFunc', kernel,...
                       'optimization', optimization);
```

以贝叶斯超参优化为例，优化过程的可视化为:





注意

- 优化算法仅支持 'bayes', 'ga', 'pso'.
- 优化参数的名字仅支持 'cost', 'degree', 'offset', 'gamma'
- 多项式核函数的参数优化只能选择 'bayes'（因为解决的是一个混合类型的优化问题）
- 多项式核函数的参数 'degree' 的类型为 'integer'

06. 交叉验证

支持 K 折交叉验证和留出法交叉验证。

例 1：5 折交叉验证

```
svddParameter = struct('cost', cost,...
                       'kernelFunc', kernel,...
                       'KFold', 5);
```

例 2：验证集比例为 0.3 的留出法交叉验证

```
svddParameter = struct('cost', cost,...
                       'kernelFunc', kernel,...
                       'Holdout', 0.3);
```

07. 基于 PCA 的数据降维

比如，将原始数据的维度降至 2 维：

```
% SVDD parameter
svddParameter = struct('cost', cost,...
                       'kernelFunc', kernel,...
                       'PCA', 2);
```

注意

仅需在 *svddParameter* 中设置降维参数即可，不需要额外的代码来处理训练数据和测试数据。

08. 加权 SVDD 模型

添加了加权 SVDD 模型的支持。

比如，添加一个数值为 '*weigh*' 的权重：

```
weight = rand(size(trainData, 1), 1);
% SVDD parameter
svddParameter = struct('cost', cost,...
                       'kernelFunc', kernel,...
                       'weight', weight);
```