libSVDD:
A Library for Support Vector Data Description

Version: 1.0.0

Sohail R. Reddy
sredd001@fiu.edu

March 16, 2020

Chapter 1

Support Vector Data Description

The support vector data description [1], also sometimes called support vector domain description, is a technique inspired by the support vector machines (SVM) of Vapnik [2] for defining an optimal description of the objects. Whereas SVM identifies a separating hyperplane that classifies objects, the SVDD approach attempts to perform the classification using hyperspheres. This method has been successfully used to define the boundary of the data set and can be used to model complex, irregular, non-convex and disconnected boundaries.

1.1 Mathematical Background

The SVDD method identifies a sphere of radius R with center $\vec{a}$ that encompasses the data set such that the radius is minimized

$$F\left(R, \vec{a}, \xi_i\right) = R^2 + C \sum_i \xi_i \tag{1.1}$$

where C represents the trade-off between the simplicity and the number of objects rejected (outliers). The radius is minimized under the following constraint

$$\|\vec{x}_i - \vec{a}\|^2 \leq R^2 + \xi_i \quad \forall_i, \xi_i \geq 0 \tag{1.2}$$

where $\vec{x}_i$ are the objects (points) whose description is being constructed. Using Eq. (1.1) and Eq. (1.2), the Lagrangian can be constructed as

$$L\left(R, \vec{a}, \alpha_i, \xi_i\right) = R^2 + C \sum_i \xi_i - \sum_i \alpha_i \left(R^2 + \xi_i - \|\vec{x}_i - \vec{a}\|^2\right) - \sum_i \gamma_i \xi_i \tag{1.3}$$

where the Lagrange multipliers satisfy $\alpha_i \geq 0$ and $\gamma_i \geq 0$. Differentiating Eq. (1.3) yields a new set of constraints

$$\begin{aligned} &\sum_i \alpha_i = 1 \quad \forall_i \\ &\vec{a} = \frac{\sum_i \alpha_i \vec{x}_i}{\sum_i \alpha_i} = \sum_i \alpha_i \vec{x}_i \quad \forall_i \\ &C - \alpha_i - \gamma_i = 0 \quad \forall_i \end{aligned} \tag{1.4}$$

It can be seen from the second constraint in Eq. (1.4) that the center of the sphere can be represented as a linear combination of weights α_i and the objects $\vec{x}_i$. Since both $\alpha_i \geq 0$ and $\gamma_i \geq 0$, the third constraint in Eq. (1.4) can be rewritten as $0 \leq \alpha_i \leq C$. Substituting Eq. (1.4) into Eq. (1.3) yields a function to maximize given by

$$L = \sum_i \alpha_i \left(\vec{x}_i \cdot \vec{x}_i\right) - \sum_{i,j} \alpha_i \alpha_j \left(\vec{x}_i \cdot \vec{x}_j\right) \tag{1.5}$$

with constraints $0 \leq \alpha_i \leq C$ and $\sum_i \alpha_i = 1$. It can be seen from the second constraint in Eq. (1.4) that only those objects $\vec{x}_i$ with $\alpha_i > 0$ are needed to define the hypersphere. These objects are referred to as support vectors. The radius of the sphere (R) can be obtained by taking the distance between the center ($\vec{a}$) and any support vector. This procedure of identifying the support vectors, α and the radius R by minimizing Eq. (1.5) is referred to as the training phrase and is similar to that used in SVM.

It is well known that an object $\vec{z}$ is on or within a sphere with the center at $\vec{a}$ and radius R if $\|\vec{z} - \vec{a}\|^2 \leq R^2$. If the center is expressed using the support vectors then the object z is on or within the sphere if

$$\left(\vec{z} \cdot \vec{z}\right) - 2\sum_i \alpha_i \left(\vec{z} \cdot \vec{x}_i\right) + \sum_{i,j} \alpha_i \alpha_j \left(\vec{x}_i \cdot \vec{x}_j\right) \leq R^2 \tag{1.6}$$

An object is "accepted" if it satisfies this acceptance criteria (Eq. 1.6) and has the same classification as the training set.

In real-world applications, the data sets are not spherically distributed and, therefore, classification using hyperspheres will not result in a tight fit (description) around the data set. This issue can be addressed by mapping the data set (objects) into a feature space where a sphere can be a better approximation. This is done using a "kernel-trick", where a kernal $K\left(\vec{x}_i \cdot \vec{x}_j\right)$ that satisfies Mercer's theorem [3] is used to map the data set. Replacing all inner products $\left(\vec{x}_i \cdot \vec{x}_j\right)$ by the kernel representation $K\left(\vec{x}_i \cdot \vec{x}_j\right)$, the data description problem is now given by

$$L = \sum_i \alpha_i K\left(\vec{x}_i \cdot \vec{x}_i\right) - \sum_{i,j} \alpha_i \alpha_j K\left(\vec{x}_i \cdot \vec{x}_j\right) \tag{1.7}$$

where constraints $0 \leq \alpha_i \leq C$ and $\sum_i \alpha_i = 1$ are enforced. Then, the acceptance criteria is given as

$$K\left(\vec{z} \cdot \vec{z}\right) - 2\sum_i \alpha_i K\left(\vec{z} \cdot \vec{x}_i\right) + \sum_{i,j} \alpha_i \alpha_j K\left(\vec{x}_i \cdot \vec{x}_j\right) \leq R^2 \tag{1.8}$$

One of the more commonly used kernel is the Gaussian kernel $K_G\left(\vec{x}_i \cdot \vec{x}_j\right) = \exp\left(-\left(\vec{x}_i - \vec{x}_j\right)^2 / \sigma^2\right)$. Using the Gaussian kernel, the data description problem can be written as

$$L = 1 - \sum_i \alpha_i^2 - \sum_{i \neq j} \alpha_i \alpha_j K_G\left(\vec{x}_i \cdot \vec{x}_j\right) \tag{1.9}$$

where the constraints are $0 \leq \alpha_i \leq C$ and $\sum_i \alpha_i = 1$. The acceptance rule then becomes

$$-2\sum_i \alpha_i K_G\left(\vec{z} \cdot \vec{x}_i\right) \leq R^2 - C_X - 1 \tag{1.10}$$

where C_X is the third term on the left-hand side of Eq. (1.8). The standard deviation of the Gaussian kernel (σ) determines the number of support vectors and the tightness of the fit, where a smaller σ results in a tighter fit and more support vectors. It should be mentioned that once the SVDD model is trained, the classification of new points in their respective regions is exceptionally fast. This is because of the acceptance criteria which simply compares the distance of the point from the center with the radius of the mapped hypersphere. The constrained minimization problem is solved using Sequential Least-Squares Quadratic Programming (SLSQP). More efficient algorithms would be stochastic gradient descent and sequential minimal optimization (SMO) but they are yet to be incorporated.

Table 1.1: Kernels available in libSVDD

Kernel Index	Name	$K\left(\vec{x} \cdot \vec{y}\right)$
1	Gaussian	$\exp\left(-\left(\lVert\vec{x} - \vec{y}\rVert/\sigma\right)^2\right)$
2	Exponential	$\exp\left(-\lVert\vec{x} - \vec{y}\rVert/\sigma^2\right)$
3	Linear	$\vec{x} \cdot \vec{y} + c_0$
4	Laplace	$\exp\left(-\lVert\vec{x} - \vec{y}\rVert/\sigma\right)$
5	Sigmoid	$\tanh\left(\gamma\left(\vec{x} \cdot \vec{y}\right) + c_0\right)$
6	Polynomial	$\left(\vec{x} \cdot \vec{y} + c_0\right)^d$

Chapter 2

Using libSVDD

2.1 Python Interface

The libSVDD library has an easy to use Python interface. The Python API requires that numpy be installed. Add the API folder to the path to access the libSVDD API.

```
sys.path.append('../../API/')
```

Import the API using

```
from libSVDD import *
```

The SVDD model for a set of labeled objects can be created using

```
svm = SVM(libSVDD = '/path/to/lib',
          objects = objects,
          labels = labels,
          sigma = 180.0,
          save = 'saveFile.dat')
```

where 'objects' is an $n \times m$ array with n objects in m dimensions, 'labels' is a vector of size n containing the label for each object. If the SVDD is being trained (rather than using a pretrained SVDD), the 'objects' and 'label' arrays must be supplied. Other parameters (their types and default values) that are possible are:

- libSVDD (string): path to the shared libSVDD.so library: (default: '')
- tol (double): the tolerance for the minimization algorithm: (default: 1e-10)
- save (string): saves the trained SVDD to the file: (default: '')

- init (string): the file from which to read a pretrained SVDD model: (default: ‘’)
- slack (double): the upper bound on the Lagrange multiplies controlling the outliers: (default: 0.25)
- kernel (integer): the kernel function (index given in Table 1.1) : (default: 1)
- sigma (double): the standard deviation in the Gaussian, Exponential and Laplace kernels: (default: 1.0)
- c0 (double): the constant c_0 in Linear, Sigmoid and Polynomial kernel: (default: 0.0)
- gamma (double): the constant γ in Sigmoid kernel: (default: 1.0)
- d (double): the exponent d in Polynomial kernel: (default: 2.0)

If the keyword ‘init’ is passed, the SVDD model can be initialized using a pretrained model as

```
svm = SVM(libSVDD = '/path/to/lib',
          init = 'initFile.dat')
```

Once the SVDD model is initialized (either trained or read), it can be used to classify a new object. This can be done using

```
radius, labels = svm.classify(test)
```

where ‘test’ is the object or set of objects to be classified, ‘radius’ is an array of type double and ‘label’ is an array of type string, both of the same size as the number of objects. If the SVDD model was trained for a multi class classification (i.e. labels of all objects were not the same), the ‘label’ will contain the label of the nearest SVDD class and ‘radius’ would contain its distance from the object. If the SVDD model was trained for single class classification, the label would indicate if the object is accepted. The object is rejected if the returned label is an empty string. The radius would then contain the distances from the center of the single SVDD class.

2.2 Fortran90 Interface

The libSVDD library was written using object-oriented Fortran90. Linking the library to a Fortran code allows for more flexible control over the library and the SVDD model.

To incorporate libSVDD into a Fortran program, import the 'support_vec_machine' module using

```
use support_vec_machine
```

Once the module is imported, create a variable of type/class 'svmmodel'

```
type(svmmodel) :: model
```

The type contains easy to use member function to train, classify, save and initialize the SVDD model. Several different parameters must be set before an SVDD model can be trained. The parameters depend on the choice of kernel function and are stored in the model%kernelParams array, where each element of the array contains a specific parameter. The arrangement of the array is shown below.

```
model%tol = 1.0d-10
model%slack = 0.25d0
model%kernelParams(1) = 1        ! Kernel index
model%kernelParams(2) = 1.0d0    ! sigma
model%kernelParams(3) = 0.0d0    ! c0
model%kernelParams(4) = 1.0d0    ! gamma
model%kernelParams(5) = 2.0d0    ! d
```

The SVDD model can then be trained using

```
call model%train(objects, labels, uniqueLabels)
```

where 'objects' is again an array of size $n \times m$, 'labels' is an array of type 'character' and size n and 'uniqueLabels' is an array of type 'character'. The size of the 'uniqueLabels' array is equal to the total number of unique labels/classes. This array contains the unique labels in the 'labels' array. The SVDD model can also be initialized using a pretrained model as

```
call model%initialize('initFile.dat')
```

The SVDD model can be saved as

```
call model%save('saveFile.dat')
```

The initialized SVDD model can now be used to classify new objects using

```
call model%classify(object, classifiedLabel,
                    radius)
```

where 'object' is an object (not a set of objects) to be classify. The variable 'classifiedLabel' is of type 'character' and contains the label of the SVDD class that the object belongs to. The variable 'radius' contains the distance to the SVDD class.

Chapter 3

Examples

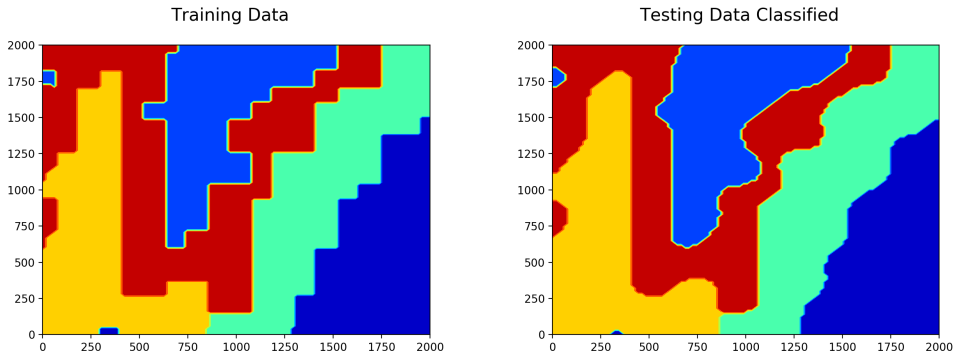
3.1 Example 1

The Example 1 is used to demonstrate how to train an SVDD model for multi-class classification. It will show how to assign labels, train the SVDD model, save the trained model to a file and use the trained model to classify new points. The source code for Example1 is given in Fortran90 ('example1.f90') and Python ('example.py'). Run the make command in the Fortran example directory to compile the examples.

The libSVDD framework is used to classify a set of points that resemble a terrain. The points on the scatted terrain is given in 'terrain.dat' file. The training data contains a total of 489 points. The extents of the terrain are $x, y \in [0, 2000]$. The terrain data is classified (assigned different labels) based on its elevation. A total of five different classes are used.

1. Label '1': $z <= 80$
2. Label '2': $z >= 230$
3. Label '3': $80 < z < 170$ and $x >= 800$
4. Label '4': $80 < z < 170$ and $x < 800$
5. Label '5': $170 <= z < 230$

An additional testing data set of 40000 points is generated to investigate the SVDD model's ability to classify the 40000 points into the appropriate regions. The testing set is given in 'testing.dat' file. Figure 3.1 shows the training set and the testing set. It can be seen that the SVDD model is able to accurately classify the terrain model.



(a) (b)

Figure 3.1: a) The training set used to train the SVDD model and b) classification of the testing set using the trained SVDD model

3.2 Example 2

The Example 2 is used to demonstrate how initialize an SVDD model from a pretrained SVDD model from Example 1. This example uses the SVDD model saved in Example 1. It will show how initialize the SVDD model from a file, save the trained model to a file and use the model to classify new points. The source code for Example1 is given in Fortran90 (‘example2.f90’) and Python (‘example.py’). Run the make command in the Fortran example directory to compile the examples.

Bibliography

[1] D. M. Tax and R. P. Duin, "Support Vector Domain Description," *Pattern Recognition Letters*, vol. 20, pp. 1191–1199, 1999.

[2] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer, 1995.

[3] N. Aronszajn, "Theory of Reproducing Kernels," *Transactions of the American Mathematical Society*, vol. 68, pp. 337–404, May 1950.