pyEDSD (SVM)

Sylvain Maugeais

24 novembre 2023

The goal of this package is to provide a python implementation of the EDSD matlab toolbox EDSD developped by Basudhar and Missoum (2008) (cf. [**basurdhar__missoum**]) and to add some features that are not present in the original.
The core of the program is in the generation of random points on the boundary of an Support Machine Vector, which is inspired but different from loc. cit. This procedure is also used for the MonteCarlo methods when estimating different invariants.
The SVM part is handled by `sklearn.svm`.

1 General algorithm

```
edsd(func, X0=[], bounds=[], N0 = 10, N1 = 10,
        processes = 1, classes = 1,svc={})
```

Given a set $\Omega \subset \mathbb{R}^d$ and a function with value in a finite set $f\colon S$, the goal is to find a description of the boundaries of the sets $\Omega_s = f^{-1}(s)$ for all $s \in S$.
First, the user must provide an initial training set X_0 containing at least one point in each region Ω_s, which is completed in a set X with N_0 points taken randomly in Ω after labelling using f. Then a first classifier `clf` is computed using this training and `sklearn.svm`, to which a dictionary containing the parameters of the `SVC` function is given.
The algorithm then adds $N1$ points using the following procedure

```
while (N1 > 0) :
        #find "processors" random points on the boundary
        P = clf.random(processes)
        #label the new points using f and add them to the
        #training set X
        X.append(P)
        # Label the new points
        y.append([f(p) for p in P])
        # Compute a new classifier using the updated training set
        clf = svm.SVC(**svc).fit(X, y)

        N1 -= processors
```

At each step, the true boundaries are refined using points on the boundary of the approximated classifier `clf`.

2 Generation of random points on the boundary

The generation of random elements on the boundary of the SVM is done using a simple conjugate gradient with respect to the square of the decision function.

Contrary to [**basurdhar__missoum**], the generated points are not required to be as far as possible form one another, as our goal is to use also this procedure for the Monte Carlo simulations.
This part can easily be parallelised, the effect of the number of parallel threads used can be found in [**lin__basurdhar__missoum**]

```
# Define the distance function as the square of the decision function
decision = lambda x : clf.decision_function([x])**2

n = 0 # Number of steps

while n < 100 :

        # Generate a random point in Omega
        x0 = random(Omega)

        # find a minimum of the distance function using the conjugate gradient
        res = minimize(decision, x0, method='CG')

        # If the result is within the bounds
        if res.x in Omega :
                return(res.x)

        n += 1
```

Although the minimisation procedure converges, the resulting point may not be in the original searching space Ω, it is therefore necessary to check if this is the case, and draw another minimisation from a new random point if necessary.

2.1 Multi-criteria

In case of multi criteria, a specific decision function is fixed randomly for each call to clf.random
and the decision function shape is taken to be 'ovo'

3 Examples

3.1 2D

```
import pyEDSD as edsd
import numpy as np
import matplotlib.pyplot as plt

def f(X) :
        r1 = np.sqrt(X[0]**2+X[1]**2)
        r2 = np.sqrt((X[0]-0.5)**2+X[1]**2)
        return((r1 > 1) or (r2 < 0.5))

if __name__ == "__main__" :
        bounds = [[-2, -2], [2, 2]]
        clf = edsd.edsd(f, X0=[[-0.5, 0], [0.5, 0], [1, 1]],
                        bounds=bounds, processes=4, classes = 2,
                        N1 = 500, svc=dict(C = 1000, gamma = 1))
        clf.draw()
```

Owing to the parallelisation, the `if __name__ == "__main__" :` is necessary for some implementation of python.

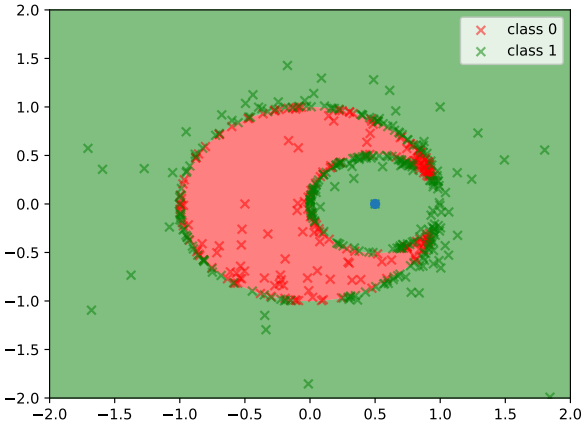

FIGURE 1 –

3.2 2D with three zones

```
import pyEDSD as edsd
import numpy as np
import matplotlib.pyplot as plt

def f(X) :
        r = np.sqrt(X[0]**2+X[1]**2)
    return(min(3, int(2*r)))

if __name__ == "__main__" :
        bounds = [[-2, -2], [2, 2]]
        clf = edsd.edsd(f, X0=[[0, 0], [0.5, 0.5], [1, 1], [1.5, 1.5]],
        N1 = 1000, svc=dict(C = 1000))
        clf.draw(grid_resolution = 1000, scatter=False)
        plt.show()
```

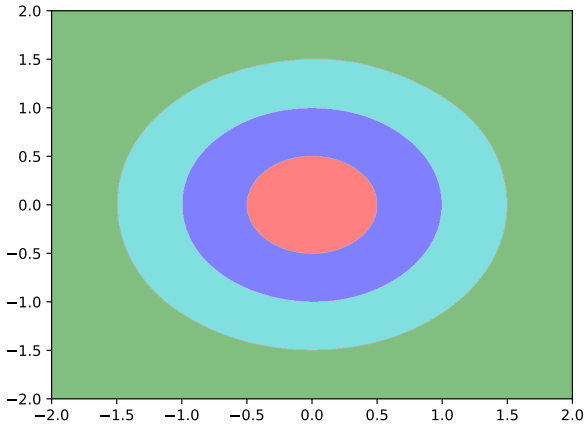

FIGURE 2 –

3.3 3D

4 Estimation of the probability distribution of the doundary

5 Estimation of invariants of the boundary

5.1 Volume in 3d

```
import pyEDSD as edsd
import numpy as np
import matplotlib.pyplot as plt


def f(X) :
        return (np.sqrt((X[0])**2+X[1]**2+(X[2])**2)>1)

if __name__ == "__main__" :
        bounds = [[-2, -2, -2], [2, 2, 2]]
        v = []

        for N1 in range(100, 5000, 100) :
                clf = edsd.edsd(f, X0=[[0, 0, 0], [1, 1, 1]], bounds=bounds,
                                processes=4, N1 = N1, svc=dict(C = 1000))
                v.append(4/3*np.pi-clf.volume(False))

        plt.semilogy(v)
        plt.grid(True)
        plt.xlabel("N1")
        plt.ylabel("Absolute error")

        plt.show()
```

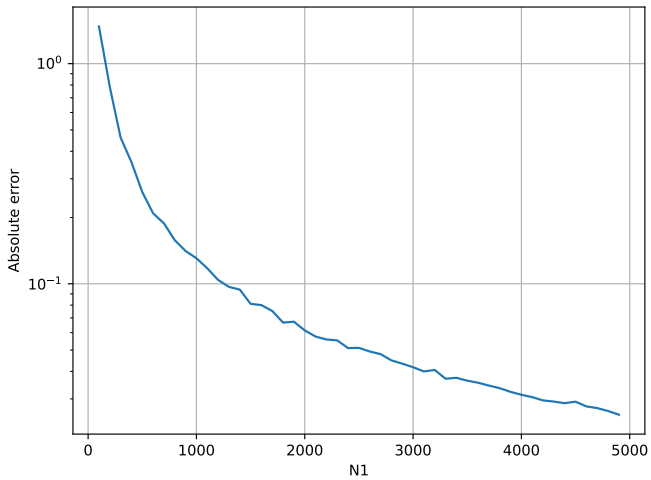

FIGURE 3 –

Basudhar, A., Missoum, S. An improved adaptive sampling scheme for the construction of explicit boundaries. Struct Multidisc Optim 42, 517529 (2010). https://-doi.org/10.1007/s00158-010-0511-0

Lin, K., Basudhar, A. and Missoum, S. (2013), "Parallel construction of explicit boundaries using support vector machines", Engineering Computations, Vol. 30 No. 1, pp. 132-148. https://doi.org/10.1108/02644401311286099