IMPLEMENTATION AND EXPERIMENTAL STUDY OF ROBUST OCSVM

Report submitted in fulfillment of the requirements
for the Exploratory Project of

Second Year B.Tech.

by

Anand Prakash (Roll No.: 18075074)
Anwoy Chatterjee (Roll No.: 18075075)
Ankit Sinha (Roll No.: 18075076)

Under the guidance of

Dr. K.K. Shukla

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY (BHU) VARANASI
Varanasi 221005, India
MAY 2020

Dedicated to

Our parents, teachers,...

Declaration

We certify that

1. The work contained in this report is original and has been done by ourselves and the general supervision of our supervisor.

2. The work has not been submitted for any project.

3. Whenever we have used materials (data, theoretical analysis, results) from other sources, we have given due credit to them by citing them in the text of the thesis and giving their details in the references.

4. Whenever we have quoted written materials from other sources, we have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Place: IIT (BHU) Varanasi

Date: 17.05.2020

Anand Prakash
Anwoy Chatterjee
Ankit Sinha
B.Tech Students
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi, Varanasi, INDIA 221005.

Certificate

*This is to certify that the work contained in this report entitled "**Implementation and Experimental Study of Robust OCSVM**" being submitted by **Anand Prakash (Roll No. 18075074), Anwoy Chatterjee (Roll No. 18075075),** and, **Ankit Sinha (Roll No. 18075076)** carried out in the Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, is a bona fide work of our supervision.*

Place: IIT (BHU) Varanasi
Date: 17.05.2020

Dr. K.K. Shukla
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi, Varanasi, INDIA 221005.

Acknowledgements

We would like to express our sincere gratitude to Dr. K.K. Shukla Sir, Dr. Ghosh and Ms. Manisha Singla Ma'am for their able guidance and support in completing the project. We would also like to thank all the Professors of our department, our parents and our friends for their constant support.

Place: IIT (BHU) Varanasi
Date: 17.05.2020

Anand Prakash
Anwoy Chatterjee
Ankit Sinha

Abstract

A Support Vector Machine (SVM) is one of the most popular Machine Learning Classifiers used nowadays and it is a supervised learning model. It is a powerful classification machine and has been applied to many fields, such as hypertext categorization , data mining and classification of images and many more, in the last few years. Although SVMs provide high accuracy and low complexity compared to other traditional models, they are associated with some problems that need to be solved. Sensitivity to outliers is one of them. One-Class Support Vector Machines (OC-SVMs), are an extension of SVMs, that reduce the effect of outliers and produce better classification results than the SVMs . In order to improve the performance even more and enhance the robustness of One-Class Support Vector Machines (OC-SVMs) we have worked on an interesting method to implement Fuzzy One-Class SVM. In this method, instead of training the One-Class Support Vector Machines (OC-SVMs) on crisp data, we have generated interval-valued training data for them to train on. Experimental results indicate that our proposed model yields better classification results compared to the traditional OC-SVMs.

Contents

Chapter 1
Introduction

1.1 Overview

Support vector machines (SVMs) is a supervised learning model which is one of the most popular classifiers. Due to its high accuracy and low complexity and high generalization ability than the many others it is widely accepted [2] . It is a powerful classification machine which is used to solve many application based problems such as hypertext categorization, data mining and classification of images, classification or recognition of fields, isolated handwritten digit recognition, object recognition, speech recognition, and spatial data analysis etc [2] . SVMs try to find the best hyperplane in the feature space that is situated at the maximum distance from both the classes. This hyperplane should be optimal so that generalization ability is maximized. The optimal hyperplane is determined with the help of a few data points which are known as support vectors(SV) [2] . Although SVM is too useful and plays an important role in dealing with many application based problems,they are also associated with some problems. Originally SVM was designed for binary classification problems. In the case of binary classification the major issues with SVMs are overfitting and sensitivity to outliers [3] .

This paper deals with this issue. In this paper the concept of fuzzification to avoid overfitting in one-class SVM is explained. In this paper the basics of fuzzy One-Class Support Vector Machines has been explained and in what terms it differs from the normal One-Class Support Vector Machines has been also discussed [4] . We have tried to generate a set of hypercubes in n-dimensional space for every training point and train our model on the fuzzy set thus obtained.It makes our model more dynamic,generic and accurate.

1.2 Motivation of the Research Work

The motivation of this project and the goal of this paper is to overcome the affore-mentioned short-comings of traditional One-Class SVMs and improve the classification accuracy. Apart from that, deeply exploring the mathematics behind SVMs,OC-SVMs and Fuzzy OC-SVMs and gaining knowledge about Fuzzy Logic have been the other driving forces to take up this research work.

1.3 Organisation of the Report

This paper is organized as follows: Chapter 2 discusses the basic theory of SVM that is classical SVM. Chapter 3 briefly explains the basics of the One-Class Support Vector Machines (OC-SVMs). In chapter 4 the concept of fuzzy One-Class Support Vector Machines (OC-SVMs) is discussed and how it is different from normal One-Class Support Vector Machines (OC-SVMs) is also explained. In chapter 5 our project work has been elaborated and also the concept of fuzzification and defuzzification is explained. Finally, in chapter 6 we have the conclusions and discussions on the results obtained.

Chapter 2
Basic Theory of SVMs

In this section, the basic mathematical formulation and classification strategy of the Support Vector Machine (SVM) is discussed. SVM is one of the most popular supervised learning methods used nowadays.

Let us assume that our training set consists of 'n' data points and out of them some are positive examples while others are negative. Let us represent the i[th] training example as x_i, where $x_i \, \varepsilon \, R^N$. SVM aims to find a decision boundary or let us say a hyperplane in such a way that the separation between the positive and negative examples is maximum. For this reason, SVM is often referred to as a 'Large Margin classifier'. The constraints it applies on the hyperplane are:

$$w^T x_i + b \geq 1 \qquad for \quad positive\ examples \tag{1}$$

$$w^T x_i + b \leq -1 \quad for \quad negative\ examples \tag{2}$$

where, w is the weight vector such that $w \, \varepsilon \, R^N$ and $\overline{w}$ is perpendicular to the separating hyperplane, and the bias b is a scalar and is actually equal to the negative of the distance of the hyperplane from the origin. To merge the inequalities (1) and (2) we introduce a new variable y_i for each x_i, such that $y_i = +1$ for positive examples, and $y_i = -1$ for negative examples. Now the inequalities (1) and (2) can be written as a single inequality, which is:

$$y_i \left(w^T x_i + b \right) \geq 1 \tag{3}$$

Here, it is worthy to note that if the inequality (3) holds for all examples in the training set, then it is said to be a linearly separable case. It can be found using simple mathematics, that the margin of separation between the two classes is $2/\|w\|$. As the basic objective of SVM is to classify the classes by maximizing their separation, so we want to maximize $2/\|w\|$, or in other words we want to minimize $\|w\|$, which is same as minimizing $\frac{1}{2}\|w\|^2$. Thus, for the linearly separable case, to find the optimal hyperplane we need to solve the following constrained optimization problem:

$$Minimize \qquad \phi(w) = \tfrac{1}{2} w^T w \tag{4}$$

$$Subject\ to \qquad y_i \left(w^T x_i + b \right) \geq 1, \quad i = 1, 2, ..., n \tag{5}$$

We can solve this constrained optimization problem using quadratic programming.

However, if the inequality (3) does not hold for all examples in the training set, then it is said to be a linearly non separable case. In such a case, the SVMs introduce a set of non-negative scalar

variables $\{\zeta_i\}_{i=1}^{n}$, to allow some training examples to violate the margin constraint. ζ_i are called slack variables. For $0 \le \zeta_i < 1$, the data points lie inside the region of separation but on the right side of the decision surface, whereas for $\zeta_i > 1$, they lie on the wrong side of the decision surface [2]. Thus, for the linearly non separable case, the margin constraint is modified as:

$$y_i \left(w^T x_i + b \right) \ge 1 - \zeta_i, \quad i = 1, 2, ..., n \tag{6}$$

SVM aims to find a separating hyperplane in such a way that there is minimum misclassification and maximum possible separation between the classes. For the linearly non separable case, to find the optimal hyperplane we need to solve the following constrained optimization problem:

$$Minimize \quad \phi(w, \zeta) = \tfrac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i \tag{7}$$

$$Subject\ to \quad y_i \left(w^T x_i + b \right) \ge 1 - \zeta_i, \quad i = 1, 2, ..., n \tag{8}$$

$$\zeta_i \ge 0, \quad i = 1, 2, ..., n \tag{9}$$

where, C is a positive parameter which is defined by the user and is called the regularization term. It controls the trade-off between minimization of misclassification error and maximization of margin of separation [2].

We can solve this constrained optimization problem by the use of Lagrange multipliers. We introduce a set of Lagrange multipliers α_i and β_i for constraints (8) and (9), and arrive at the Lagrangian:

$$L(w, \zeta, b, \alpha, \beta) = \tfrac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i - \sum_{i=1}^{n} \alpha_i \left(y_i \left(w^T x_i + b \right) - 1 + \zeta_i \right) - \sum_{i=1}^{n} \beta_i \zeta_i \tag{10}$$

Hence, the dual problem is:

$$Maximize \quad Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{11}$$

$$Subject\ to \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{12}$$

$$0 \le \alpha_i \le C, \quad i = 1, 2, ..., n \tag{13}$$

It can be observed that in the dual problem, the dependence of the objective function $Q(\alpha)$ on the data points in training set is only in the form of a set of dot product: $\{x_i^T x_j\}_{(i,j)=1}^{n}$. In the process of finding the saddle point of the Lagrangian L, we get $\beta_i = C - \alpha_i$. We also observe that the α_i s are of two kinds, and of them the data points corresponding to $0 < \alpha_i \le C$ are called the Support Vectors (SVs). We get the optimal weight vector to be:

$$w_o = \sum_{i=1}^{N_s} a_i y_i x_i \tag{14}$$

where, N_s is the number of SVs.The optimal bias b_o can be found by considering any data point from the training set for which we have $0 < \alpha_i < C$ (and so, $\zeta_i = 0$) and using the data point in the equation resulting from the Kuhn-Tucker condition on inequality (8). However, for a

better result we take the mean value of b_o obtained for all data points in training set having $0 < \alpha_i < C$. Now as we have the optimal pair (w_o, b_o), we can write our decision function as:

$$g(x) = sgn\left(\sum_{i=1}^{N_s} \alpha_i y_i x_i^T x + b_o\right) \tag{15}$$

Hence, the equation of the separating hyperplane is $g(x) = 0$.

Now, for the non linear case, we simply map the data into an alternative higher dimensional feature space through a non linear mapping function $\psi(x) \varepsilon R^M$, where $M > N$. The data points become linearly separable in this new feature space and thus, the SVM can easily find the separating hyperplane in this space. The feature map ψ is endowed with the inner-product kernel $K(x, x_i)$ [1], where,

$$K(x, x_i) = K(x_i, x) = \psi(x)^T \psi(x_i) \tag{16}$$

Thus, the corresponding dual optimization problem for this case is:

$$Maximize \quad Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K\left(x_i, x_j\right) \tag{17}$$

subject to the constraints (12) and (13). It must be noted that $K(x, x_i)$ is a kernel which satisfies the Mercer's theorem. Using Kernel functions, the decision rule is:

$$\begin{aligned} x \varepsilon \quad & positive\ class,\ if\ g(x) > 0 \\ & negative\ class,\ if\ g(x) < 0 \end{aligned} \tag{18}$$

where, $g(x)$ is the decision function and is defined as:

$$g(x) = sgn\left(\sum_{SVs} \alpha_i y_i K(x, x_i)\right) \tag{19}$$

Chapter 3
One-class SVM

The idea of the One-Class SVM as developed by Scholkopf was to treat all the training examples to belong to one class and the origin to be the only member of the other class. Let $x_1, x_2, x_3, x_4, x_5, ..., x_n \, \varepsilon \, R^m$ be the data points in the R^m space.Let $K : R^m \rightarrow C$ be the kernel function that transforms the input space to another space.Given a probability distribution P in the new feature space we have to find a subset Q such that the probability that a test example lies outside Q is bounded by a pre-specified value in the interval (0,1).The algorithm separates all the training points from the origin and maximizes the distance of the hyperplane from the origin. The decision function is a binary-valued function. The function returns +1 in the region Q (within which the training points lie) and −1 in the region outside.

To find out the required hyperplane we need to solve the quadratic programming minimization:

$$min_{w,\xi i,\rho} \; \tfrac{1}{2}\|w\|^2 \; + \; \tfrac{1}{\nu n} \; \sum_{i=1}^{n} \xi_i \; - \; \rho \tag{20}$$

$$\text{subject to: } (w \cdot \phi(x_i)) \geq \rho - \xi_i \;\; , \; \xi_i \geq 0 \;\; \text{for all } \; i=1,\ldots,n \tag{21}$$

In the above formula the parameter ν plays a significant role:

1. It sets an upper-bound on the fraction of outliers.
2. It sets a lower bound on the number of training examples used as Support Vectors.

If w and ρ solve the equation then the resulting decision function is:

$$f(x) = \text{sgn}((w \cdot \phi(x_i) - \rho) = sgn\left(\sum_{i=1}^{N} \alpha_i K(x, x_i) - \rho\right) \tag{22}$$

Chapter 4
Fuzzy One-class SVM

In many real-world applications, it happens that the influence of each training point differs .All the training points do not represent the class equally. Some training points prove to be more important than others in the problem concerned. In that scenario our priority is to correctly classify the meaningful training points and not be bothered by whether our model classifies the outliers correctly or not. So, each training point no more belongs to the positive class only. It may 70% belong to the positive class and 30% be meaningless.

So,we can associate a fuzzy membership value μ_i to each training point x_i such that $0 < \mu_i \leq 1$ .This value signifies the relative importance of the point to the target class. While,the value $(1 - \mu_i)$ represents the point's degree of belonging outside of the target class.By this approach we are able to avoid overfitting in one-class SVM. In this way during the training process the model will learn the data points with different importance.

The constrained optimization problem for the fuzzy one-class SVM is formulated as:

$$min_{w,b,\xi_i} \quad \tfrac{1}{2}\| w \|^2 + C \sum_{i=1}^{N} \mu_i \xi_i - \rho \tag{23}$$

$$subject\ to \ \ (w \bullet \phi(x_i)) \geq \rho - \xi_i , \quad \xi_i \geq 0 , \ \forall\ i = 1, 2, 3, 4, 5, ..., N. \tag{24}$$

By the method of Lagrange Multipliers we can solve this optimisation problem in dual variables as:

$$L = \tfrac{1}{2}\| w \|^2 + C \sum_{i=1}^{N} \mu_i \xi_i - \rho - \sum_{i=1}^{N} \alpha_i ((w \bullet \phi(x_i)) - \rho + \xi_i) - \sum_{i=1}^{N} \beta_i \xi_i \tag{25}$$

Here, α_i and β_i are the nonnegative Lagrange multipliers. Differentiating L with respect to w, b and ξ_i and after equating them to zero, the equations are:

$$\partial L/\partial w = w - \sum_{i=1}^{N} \alpha_i \varphi(x_i) = 0 \Rightarrow w = \sum_{i=1}^{N} \alpha_i \varphi(x_i), \quad (26)$$

$$\partial L/\partial b = 1 - \sum_{i=1}^{N} \alpha_i = 0 \Rightarrow \sum_{i=1}^{N} \alpha_i = 1, \quad (27)$$

$$\partial L/\partial \xi_i = C\mu_i - \alpha_i - \beta_i = 0 \Rightarrow \alpha_i = C\mu_i - \beta_i \ \textit{and} \ \alpha_i \le C\mu_i. \quad (28)$$

Substituting the equations (26),(27) and (28) into equation (25), the Lagrangian is a function of α only. The dual problem becomes:

$$min_{\alpha_i} \quad 1/2 * \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j k(x_i, x_j)$$

$$\text{subject to} \quad 0 \le \alpha_i \le C\mu_i, \quad \forall i = 1, 2, 3, 4, 5, ..., N, \quad \sum_{i=1}^{N} \alpha_i = 1. \quad (29)$$

The Fuzzy one-class SVM differs from normal one-class SVM only in the upper bound of Lagrange multiplier α_i corresponding to each training point x_i.

Chapter 5
Project Work

In our project we have implemented a Fuzzy One-Class SVM model. The fuzzy rule base that we have used is: for each training point $(X^{(i)}, y)$ where $X^{(i)} = (x_1, x_2, x_3, ..., x_m)$ we have created points $X'^{(i)(j)} = (x'_1, x'_2, x'_3, ..., x'_m)$ such that $x_i - \delta * x_i \leq x'_i \leq x_i + \Delta * x_i$. Here, δ and Δ are pre-specified fractions. We have used a square membership function:

$\mu(x) = \alpha, \ |x - c| \leq l(1 - \alpha)$

$= 0, \ |x - c| > l(1 - \alpha)$

such that c and l are the center and the range of the bounds on x'_i.

Let $lb = x_i - \delta * x_i$ *and*

$ub = x_i + \Delta * x_i$ *then*

$c = (lb + ub)/2$ *and* $l = (ub - lb)/2$.

Here, α is the membership value and it lies in the range $0 < \alpha \leq 1$ .Further, we have defined a set of α values uniformly distributed between (0,1] into a specified number of intervals.The picture that forms in the n-dimensional space is that every training point is enclosed by a set of hypercubes.All the points lying on a particular hypercube share the same membership value.The farther the points are from the training point the lower are their membership value.We have trained our model on the fuzzy set thus obtained from the given crisp dataset.The resulting learned parameters are also fuzzy sets of their own. Then, we have defuzzified those parameters to crisp parameters.The final crisp parameters were then used to make predictions on the test set.

Fuzzification

It is the method of transforming a crisp quantity into a fuzzy quantity. This can be achieved by identifying the various known crisp and deterministic quantities as completely nondeterministic and quite uncertain in nature. This uncertainty may have emerged because of vagueness and

imprecision which then lead the variables to be represented by a membership function as they can be fuzzy in nature.

Algorithm:

1. We have created a set of values called the alpha set from the interval (0,1) such that each alpha value is equidistant from the others. For each alpha value we have done the following:
2. For each dimension x_i of the crisp point $(x_1, x_2, x_3, \ldots, x_m)$ we have fuzzified it as:

   $([\, x_i - g * x_i \,,\, x_i + f * x_i \,])$

   $LBi = x_i - g * x_i$

   $UBi = x_i + f * x_i$

3. For each dimension (co-ordinate) x_i we have calculated centre and range as $(UBi+LBi)/2$ and $(UBi-LBi)/2$ respectively.
4. For every point we have done the following step 5.
5. Corresponding to each dimension we did the following for every bound of it:

   a. Fixing that bound as X_b we have generated a point

   $(X_1, X_2, X_3, \ldots, X_m)$

   such that X_i is chosen from the uniform distribution :

   $f(x_i) = 1, \quad LB_i < x_i < UB_i$

   $= 0, \quad x_i \in (-\infty, LB_i] \cup [UB_i, +\infty)$

   Hence, for each point we have generated 2*n fuzzified points(n = number of dimensions).

6. We have inserted those fuzzy points in a new training set.
7. Then, we have trained our model on the new training set.
8. We appended the parameters/coefficients of our trained model into lists of the particular type.

Defuzzification

It is the process of converting fuzzy data into crisp data with respect to the fuzzy set.It is a sort of approximation, where a fuzzy set having a group of membership values on the unit interval is

reduced to a single scalar value. In a Fuzzy Logic Controller (FLC) the defuzzified output signifies the action to be taken to control the process under consideration.

The different methods of defuzzification are as follows:

1. Center of Sums Method (COS)
2. Center of gravity (COG) / Centroid of Area (COA)
3. Method Center of Area / Bisector of Area Method (BOA)
4. Weighted Average Method
5. Maxima Methods
    a. First of Maxima Method (FOM)
    b. Last of Maxima Method (LOM)
    c. Mean of Maxima Method (MOM)

Algorithm:

1. We have a weight vector for each alpha value as maintained by the separate lists. For each list of coefficients we do the following steps:
2. For each coefficient in the list we map it to the corresponding **maximum alpha(α) value.**
3. Now, we create another field as alpha*alpha*coefficient i.e. ($\alpha i * \alpha i * \theta i$) .
4. We defuzzify each coefficient as:

$$\overline{\theta} = \left(\int_{\alpha\ initial}^{\alpha\ final} \alpha^2 * \theta\ d\alpha \right) / \left(\int_{\alpha\ initial}^{\alpha\ final} d\alpha \right) = \frac{\int_{\alpha\ initial}^{\alpha\ final} \alpha^2 \theta d\alpha}{\int_{\alpha\ initial}^{\alpha\ final} d\alpha}$$

5. After obtaining the defuzzified parameters we use them to predict on the test set.

Chapter 6
Conclusions and Discussion

We have implemented our model in Python language and at first tested it on a linearly separable synthetic dataset. Figure-1 below shows the fuzzified data points and the decision boundaries we got for different alpha values.

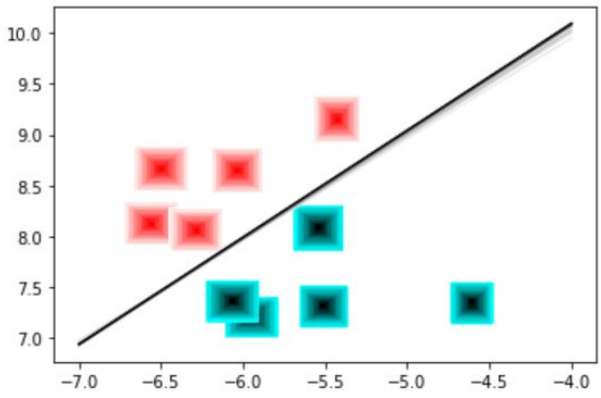

Fig. 1. The fuzzified data points with the decision boundaries
for different alpha values, for the linearly separable synthetic dataset

To get an idea of our model's performance relative to the well-established Classical OC-SVM model , we have tried our model on a synthetic dataset and a real world dataset.
We generated a non linear synthetic dataset using the *make_moons* function available in the *scikit-learn* library, and found that our model gave better results compared to the Classical One-Class SVM model which uses crisp data points for training. The comparison is summarised in Table-1 below:

SYNTHETIC DATASET (number of samples=30)

Fuzzy One-Class SVM (accuracy)	Classical One-Class SVM (accuracy)
0.80	0.20

Table 1

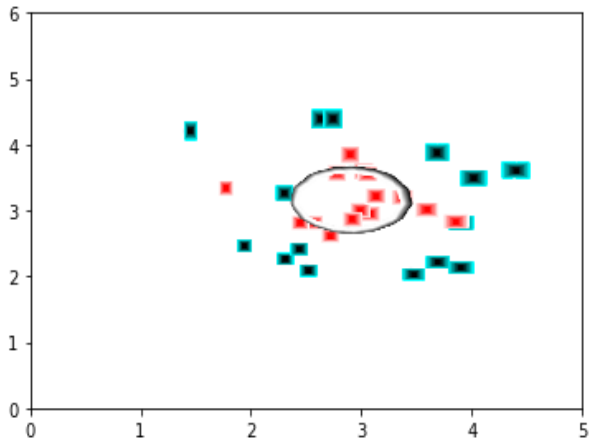

Fig. 2. The fuzzified data points with the decision boundaries for different alpha values, for the non linear synthetic dataset.

We also tried our model on the popular 'Iris Dataset' whose results are summarised in Table-2.

IRIS DATASET (number of samples=150)

Species	Fuzzy One-Class SVM (accuracy)	Classical One-Class SVM (accuracy)
0	0.84	0.15333333333333332
1	0.70	0.16666666666666666
2	0.72	0.16666666666666666

Table 2

So, it is evident from the result that our model has performed better than the normal One-Class SVM classifier on both the datasets.The main reason behind it is that our model had not only learned from the training points but also from the neighbourhood of those points with varying membership values.So,the data fed to our model becomes much more due to fuzzification as compared to the case of classical One-Class SVM. So far, we have only worked with Linear and Gaussian(rbf) kernels. In the future there is scope to work with other kernels. We have also been able to fuzzify points in n-dimensions. However, we are yet to prepare our model to train and test on n-dimensional feature space.

Bibliography

[1] Lev V.Utkin and Yulia A. Zhuk, “Knowledge-Based Systems ”120(2017), pp.43-56.

[2] H.P.Huang and Yi-Hung Liu,“International Journal of Fuzzy Systems ” Vol.4, No.3, september2002 , pp. 826-834.

[3] Y.Tian “Neurocomputing”000(2018), pp.1-13.

[4] Pei-Yi Hao, “Fuzzy one-class support vector machines” Fuzzy Sets and Systems 159 (2008) 2317 – 2336.

[5] Larry M. Manevitz and Malik Yousef, “One-Class SVMs for Document Classification” Journal of Machine Learning Research 2 (2001) 139-154.

[6] Thirumalai Muthu Thirumalaiappan Ramanathan and Dharmendra Sharma, “An SVM-Fuzzy Expert System Design For Diabetes Risk Classification” (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (3) , 2015, 2221-2226.

[7] Yingjie Tian, Yong Shi and Xiaohui Liu, “RECENT ADVANCES ON SUPPORT VECTOR MACHINES RESEARCH” TECHNOLOGICAL AND ECONOMIC DEVELOPMENT OF ECONOMY 2012 Volume 18(1): 5–33.

Appendix

```
#!/usr/bin/env python
# coding: utf-8

# In[73]:


import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs,make_moons,make_circles
from sklearn.model_selection import train_test_split
from sklearn.svm import OneClassSVM as OCSVM
from sklearn.gaussian_process.kernels import RBF
from sklearn.svm import SVC
from sklearn import datasets


# In[54]:


X, Y = make_circles(n_samples=30,factor=0.1,noise=0.4,random_state=5)
X=X+3


# In[55]:


X1=[]
Y1=[]
for i in range(len(X)):
    if Y[i]==1:
        X1.append(X[i])
        Y1.append(Y[i])
X1=np.array(X1)
Y1=np.array(Y1)
```

```
# In[56]:


plt.scatter(X[:,0],X[:,1],c=Y)


# In[57]:


# plot the decision function
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
xx = np.linspace(0, 5, 30)
yy = np.linspace(0, 6, 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T


# In[86]:


class FzOCSVM:
    def __init__(self,n_alpha,kernel):
        self.n_alpha=n_alpha
        self.kernel=kernel
        self.alpha=np.linspace(0,1,self.n_alpha)
        self.fuzz_points=[]
        self.ffy=[]
        self.fdata=[]
        self.fY=[]
        self.cff_list=[]
        self.rho_list=[]
        self.weight=[]
        self.bias=[]
        self.clf2=OCSVM(kernel=self.kernel)
        self.SV_list=[]
        self.w=[]
        self.b=[]
        self.fw=[]
        self.fb=[]

    def Gen_lb_ub(self,x):
```

```
    lb=x-0.02*x
    ub=x+0.03*x
    return lb,ub

def Gen_cen_ran(self,lb,ub):
    cen=(ub+lb)/2
    ran=(ub-lb)/2
    return cen,ran

def Fuzzify(self,cen,ran,alpha):
    a=cen-ran*(1-alpha)
    b=cen+ran*(1-alpha)
    return a,b

def Gen_setx(self,a,b,c,y,ffy):
    #print("in gen set x")
    for i in range(40):
        self.fuzz_points.append([np.random.uniform(a,b),c])
        self.fY.append(y)
        #self.ffy.append(y)
    self.fuzz_points.append([a,c])
    self.fuzz_points.append([b,c])
    self.fY.append(y)
    self.fY.append(y)
    #print(self.fuzz_points)
    #self.ffy.append(y)
    #self.ffy.append(y)

def Gen_sety(self,a,b,c,y,ffy):
    for i in range(40):
        self.fuzz_points.append([c,np.random.uniform(a,b)])
        self.fY.append(y)
        #self.ffy.append(y)
    self.fuzz_points.append([c,a])
    self.fuzz_points.append([c,b])
    self.fY.append(y)
    self.fY.append(y)
    #self.ffy.append(y)
    #self.ffy.append(y)
#function to plot the data points for each alpha
def plot_data(self,x,y):
    lb,ub=self.Gen_lb_ub(x)
    cen,ran=self.Gen_cen_ran(lb,ub)
```

```
        for i in range(11):
            #global fuzz_points
            self.fuzz_points=[]
            a,b=self.Fuzzify(cen,ran,self.alpha[i])
            self.Gen_setx(a[0],b[0],a[1],y,self.ffy)
            self.Gen_setx(a[0],b[0],b[1],y,self.ffy)
            self.Gen_sety(a[1],b[1],a[0],y,self.ffy)
            self.Gen_sety(a[1],b[1],b[0],y,self.ffy)
            self.fdata.extend(self.fuzz_points)
            #print(len(fdata))
            fuzz_points_new=np.array(self.fuzz_points)
            #print(fuzz_points_new[:,:])
            t=(y,1-self.alpha[i],1-self.alpha[i])
            #plt.scatter(i,i,color=(0,alpha[i],alpha[i]))
            plt.scatter(fuzz_points_new[:,0],fuzz_points_new[:,1],color=t,s=2,cmap='coolwarm')
            #print(alpha[i])

                                    def          plot_svc_decision_function(self,model,X,Y,          ax=None,
plot_support=True,alpha_value=0.7):
        """Plot the decision function for a 2D SVC"""
        plt.scatter(X[:, 0], X[:, 1], c=Y, s=30, cmap=plt.cm.Paired)

        #plot the decision function
        ax = plt.gca()
        xlim = ax.get_xlim()
        ylim = ax.get_ylim()

        #create grid to evaluate model

        Z = model.decision_function(xy).reshape(XX.shape)

        #plot decision boundary and margins
            ax.contour(XX, YY, Z, """colors=(alpha_value,alpha_value,alpha_value)""", levels=[-1, 0,
1], alpha=1,
                    linestyles=['--', '-', '--'])
        #plot support vectors
        #ax.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1], s=100,
        #           linewidth=1, facecolors='none', edgecolors='k',alpha=1-alpha_value)
        #plt.show()

    def predict(self,X,Y,index):
        rbf=RBF()
        SV=self.SV_list[index]
```

```
    val=0
    distances=[]
    y_pred=[]
    n=len(X)
    m=len(SV)
    for i in range(n):
        val=0
        for j in range(m):
            B=[]
            C=[]
            B.append(SV[j])
            C.append(X[i])
            result=rbf.__call__(B,C)
            #print(result)
            val=val+self.cff_list[index][j]*result[0][0]
        val=val+self.rho_list[index]
        distances.append(val)
        if val>0:
            y_pred.append(1)
        elif val<0:
            y_pred.append(0)
    y_pred=np.array(y_pred)
    return y_pred,distances

def main(self,X,Y,n_samples):
    for i in range(self.n_alpha):
        #global fuzz_points
        self.fuzz_points=[]
        #global ffy
        self.ffy=[]
        t=[]
        for j in range(n_samples):
            lb,ub=self.Gen_lb_ub(X[j])
            cen,ran=self.Gen_cen_ran(lb,ub)
            a,b=self.Fuzzify(cen,ran,self.alpha[i])
            self.Gen_setx(a[0],b[0],a[1],Y[j],self.ffy)
            #print(self.fuzz_points,j)
            self.Gen_setx(a[0],b[0],b[1],Y[j],self.ffy)
            self.Gen_sety(a[1],b[1],a[0],Y[j],self.ffy)
            self.Gen_sety(a[1],b[1],b[0],Y[j],self.ffy)
            #print(self.fuzz_points)
            self.fdata.extend(self.fuzz_points)
            #x1=np.linspace(2,5)
```

```
            self.fuzz_points_new=np.array(self.fuzz_points)

#X_train,X_test,Y_train,Y_test=train_test_split(fuzz_points,ffy,test_size=0.3,random_state=1)
            self.clf2.fit(self.fuzz_points)
            cff=self.clf2.dual_coef_[0]
            rho=self.clf2.intercept_[0]
            sv=self.clf2.support_vectors_
            self.cff_list.append(cff)
            self.rho_list.append(rho)
            self.SV_list.append(sv)
            #y_pred,distances=self.predict(X,Y,i)
            #print((y_pred==Y).mean())
            #m=-w[0][0]/w[0][1]
            #x2=m*x1-b[0]/w[0][1]
            #SV=clf2.support_vectors_
            #x2=0
            self.plot_svc_decision_function(self.clf2,X,Y,alpha_value=self.alpha[i])
        self.deFuzzify()
        #plt.scatter(X[0,:],X[1,:],c=Y)
        #plt.show()
            #for i in range(len(SV)):
            #x2=x2+cff[i]*rbf.__call__(SV[i],)
            #plt.plot(x1,x2,c=(1-alpha[i],1-alpha[i],1-alpha[i]),linewidth=1.0)

#plt.scatter(SV[:,0],SV[:,1],s=300,lw=1,c=(1-alpha[i],1-alpha[i],1-alpha[i]),facecolors='none')
            #print(SV)
    def deFuzzify(self):
        n=min(list(map(len,self.cff_list)))
        for j in range(n):
            d={}
            for i in range(len(self.cff_list)):
                #if self.cff_list[i][j] not in d.keys():
                d[self.cff_list[i][j]]=[]
            for i in range(len(self.cff_list)):
                d[self.cff_list[i][j]].append(self.alpha[i])
            self.w=[]
            self.fw=[]
            for (key,value) in d.items():
                maxalpha=max(value)
                self.fw.append(maxalpha)
                self.w.append(key)
            output=np.trapz(y=self.fw,x=self.w)
            self.weight.append(output)
```

```
        r={}
        for i in range(len(self.rho_list)):
            if self.rho_list[i] not in r:
                r[self.rho_list[i]]=[]
        for i in range(len(self.rho_list)):
            r[self.rho_list[i]].append(self.alpha[i])
        self.b=[]
        self.fb=[]
        for (key,value) in r.items():
            maxalpha=max(value)
            self.fb.append(maxalpha)
            self.b.append(key)
        output=np.trapz(y=self.fb,x=self.b)
        self.bias=output


# In[63]:


fob=FzOCSVM(n_alpha=11,kernel='rbf')


# In[64]:


fob.main(X1,Y1,len(X1))
plt.scatter(X[:, 0], X[:, 1], c=Y, s=30, cmap=plt.cm.Paired)
for i in range(30):
    fob.plot_data(X[i],Y[i])
plt.show()


# In[65]:


y_pred,distances=fob.predict(X,Y,2)


# In[66]:


print((y_pred==Y).mean())
```

```
# In[69]:


clf1=OCSVM()
clf1.fit(X1)
y_pred1=clf1.predict(X)
print((y_pred1==Y).mean())


# In[32]:


print(len(fob.SV_list[6]))
print(len(fob.weight))


# In[ ]:


def plot_svc_decision_function(model,X,Y, ax=None, plot_support=True,alpha_value=0.7):
        """Plot the decision function for a 2D SVC"""
        plt.scatter(X[:, 0], X[:, 1], c=Y, s=30, cmap=plt.cm.Paired)

        #plot the decision function
        ax = plt.gca()
        xlim = ax.get_xlim()
        ylim = ax.get_ylim()

        #create grid to evaluate model

        Z = model.decision_function(xy).reshape(XX.shape)

        #plot decision boundary and margins
           ax.contour(XX, YY, Z, """colors=(alpha_value,alpha_value,alpha_value)""", levels=[-1, 0,
1], alpha=1,
                   linestyles=['--', '-', '--'])
        #plot support vectors
        ax.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1], s=100,
                   linewidth=1, facecolors='none', edgecolors='k',alpha=1-alpha_value)
        plt.show()
```

```
# In[ ]:


#cnt=0
#for SV in fob.SV_list:
#    print(cnt)
#    for i in range(len(SV)):
#        plt.scatter([SV[i][0]],[SV[i][1]])
#    cnt=cnt+1
#plt.show()


# In[ ]:


k=min(list(map(len,fob.SV_list)))
print(k)
s=fob.alpha.sum()
n=len(fob.SV_list)
defzSV=[]
for j in range(k):
    arr=np.array([0,0])
    arr2=np.array([0,0])
    for i in range(n):
        arr=arr+fob.SV_list[i][j]
        arr2=arr2+fob.alpha[i]*fob.SV_list[i][j]
        #plt.scatter([arr[0]],[arr[1]])
    arr=arr/n
    arr2=arr2/s
    defzSV.append(arr2)
    plt.scatter([arr[0]],[arr[1]])
    #plt.scatter([arr2[0]],[arr2[1]])
plt.show()


# In[45]:


print(len(fob.fdata))


# In[ ]:
```

```
#fob.clf2.fit(fob.fdata)
plot_svc_decision_function(fob.clf2,X,Y)


# In[74]:


def pre_process(X,Y,cls):
    X1=[]
    Y1=[]
    for i in range(len(X)):
        if Y[i]==cls:
            X1.append(X[i])
            Y1.append(Y[i])
    X1=np.array(X1)
    Y1=np.array(Y1)
    return X1,Y1


# In[102]:


iris=datasets.load_iris()
X2,Y2=pre_process(iris.data[:,:2],iris.target,2)


# In[109]:



x_min, x_max = iris.data[:, 0].min() - .5, iris.data[:, 0].max() + .5
y_min, y_max = iris.data[:, 1].min() - .5, iris.data[:, 1].max() + .5
fob2=FzOCSVM(n_alpha=11,kernel='rbf')
fob2.main(X2,Y2,len(X2))
for i in range(len(iris.data[:,0])):
    fob2.plot_data(iris.data[i,:2],iris.target[i]/2)

plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(0, 10)
plt.ylim(0, 10)
```

```
plt.xticks(())
plt.yticks(())

plt.show()


# In[104]:


def evaluate(cls,y_pred):
    cnt=0
    n=len(iris.data[:,0])
    for i in range(len(iris.data[:,0])):
        if iris.target[i] == cls and y_pred[i]==1:
            cnt=cnt+1
        elif iris.target[i] != cls and y_pred[i]==0:
            cnt=cnt+1
    return cnt/n


# In[106]:


y_pred2,distances2=fob2.predict(iris.data[:,:2],iris.target,1)
result=evaluate(2,y_pred2)
print(result)


# In[97]:


print(iris.target)


# In[115]:


X3,Y3=pre_process(iris.data[:,:2],iris.target,2)
clf3=OCSVM(kernel='rbf')
clf3.fit(X3)
y_pred3=clf3.predict(iris.data[:,:2])
result3=evaluate(2,y_pred3)
print(result3)
```

In[116]:

```
print(len(iris.data[:,0]))
print(len(iris.data))
```