IMPLEMENTATION AND EXPERIMENTAL STUDY OF ROBUST OCSVM

Submitted by:

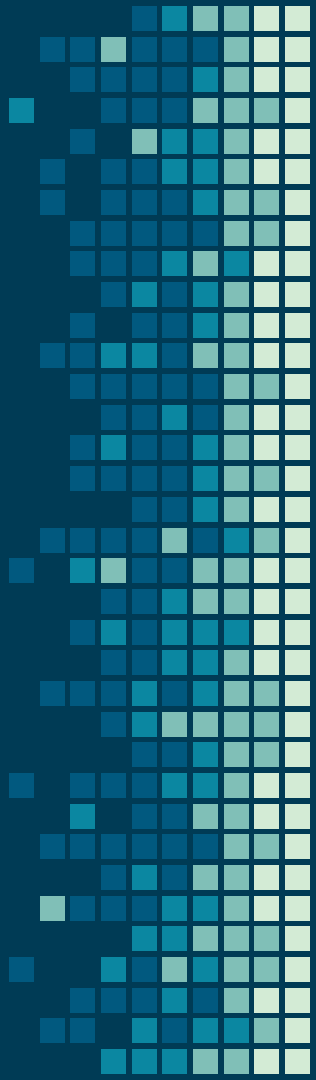
- ★ Anand Prakash (Roll No.: 18075074)
- ★ Anwoy Chatterjee (Roll No.: 18075075)
- ★ Ankit Sinha (Roll No.: 18075076)

Supervisor: Dr. K.K. Shukla

INTRODUCTION

A Support Vector Machine (SVM) is one of the most popular Machine Learning Classifiers used nowadays as it provides high accuracy and low complexity compared to other traditional models. But SVMs are associated with some problems, like sensitivity to outliers and overfitting. 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 enhance the robustness of One-Class Support Vector Machines (OC-SVMs) we have worked on an interesting method to implement Fuzzy One-Class SVM. We have used the concept of fuzzification to avoid overfitting in one-class SVM and to make it more robust. We have tried to generate a set of n-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. Experimental results also indicate that our proposed model yields better classification results compared to the traditional OC-SVMs.

FUZZY ONE-CLASS SVM



WHY FUZZY OC-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. Though the traditional One-Class SVM (OC-SVM) is considered to be an effective classification technique for anomaly detection problems, it suffers from the problem of sensitivity to the presence of outliers and noises in the training sets. This problem of high sensitivity to outliers and noises arises mainly because all training data points are equally treated during the training of traditional OC-SVMs. So, to solve this problem in Fuzzy OC-SVM, we assign each data point a membership value according to its relative importance in the class. So, each training point no more belongs to the positive class(or, negative class) only. For example, it may 70% belong to the positive class and 30% be meaningless.

MATHEMATICAL FORMULATION

We associate a fuzzy membership value μ_i to each training point x_i such that $0 < \mu_i \leq 1$. This value μ_i signifies the relative importance of the point to the target class, while the value $(1\text{-}\mu_i)$ 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:

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

$$subject\ to \quad (w.\phi(x_i)) \geq \rho - \xi_i\,, \qquad \xi_i \geq 0, \qquad \forall i = 1,2,3,\dots,N$$

MATHEMATICAL FORMULATION

Using the method of Lagrange multipliers, the optimization problem can be reduced to the following dual problem:

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

$$subject\ to\ \ 0 \le \alpha_i \le C\mu_i, \qquad \forall i = 1,2,3,\dots,N, \qquad \sum_{i=1}^{N}\alpha_i = 1$$

where, α_i are the non-negative Lagrange multipliers, and $K(x,x_i)$ is a kernel which satisfies the Mercer's theorem.

PROJECT WORK

OUR APPROACH

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 $X^{(i)} = (x_1, x_2, x_3, \dots, x_m)$ we have created points $X'^{(i)(j)} = (x'_1, x'_2, x'_3, \dots, x'_m)$, 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:

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

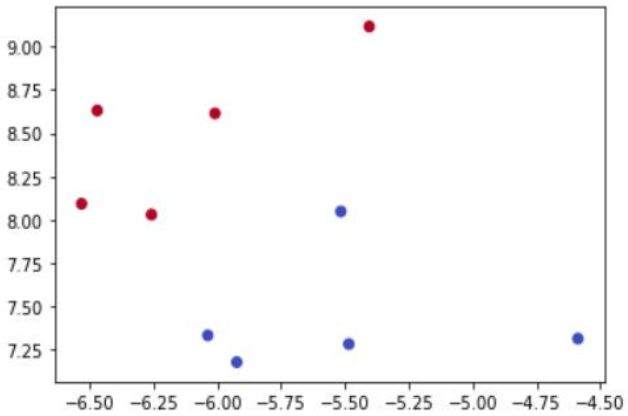
such that, c and l are the center and the range of the bounds on x'_i. Let $lb = (x_i - \delta * x_i)$ and $ub = (x_i + \Delta * x_i)$, then, $c = (lb + ub)/2$ and $l = (ub - lb)/2$. Here, α is the membership value and it lies in the range $0 < \alpha \leq 1$. Further, we have defined a set of α values uniformly distributed between $(0,1]$ into a specified number of intervals.

OUR APPROACH

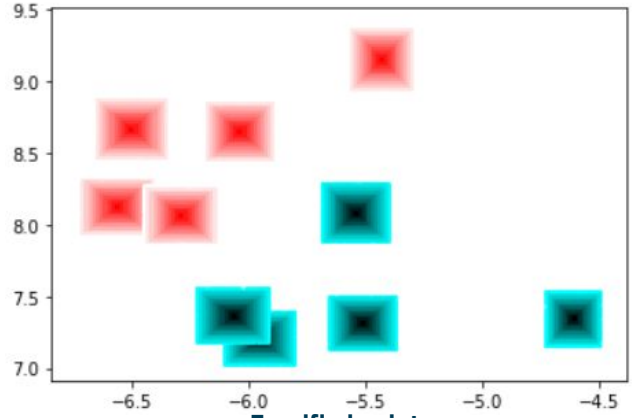
- The picture that forms in the n-dimensional space is that every training point is enclosed by a set of n-hypercubes. All the points lying on a particular hypercube share the same membership value. The farther the points are from the actual training point, the lower are their membership value.
- We 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 non-deterministic 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.



Crisp points



Fuzzified points

Each point has a degree of presence in its neighbourhood

FUZZIFICATION 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 $(x_1, x_2, x_3, \dots, x_m)$ we have fuzzified it as: $([x_i - g * x_i, x_i + f * x_i])$

   $LB_i = x_i - g*x_i$

   $UB_i = x_i + f*x_i$
3. For each dimension (co-ordinate) x_i we have calculated centre and range as $(UB_i + LB_i)/2$ and $(UB_i - LB_i)/2$ 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:

FUZZIFICATION ALGORITHM

5.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 :

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

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

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:

- Center of Sums Method (COS)
- Center of gravity (COG) / Centroid of Area (COA)
- Method Center of Area / Bisector of Area Method (BOA)
- Weighted Average Method
- Maxima Methods

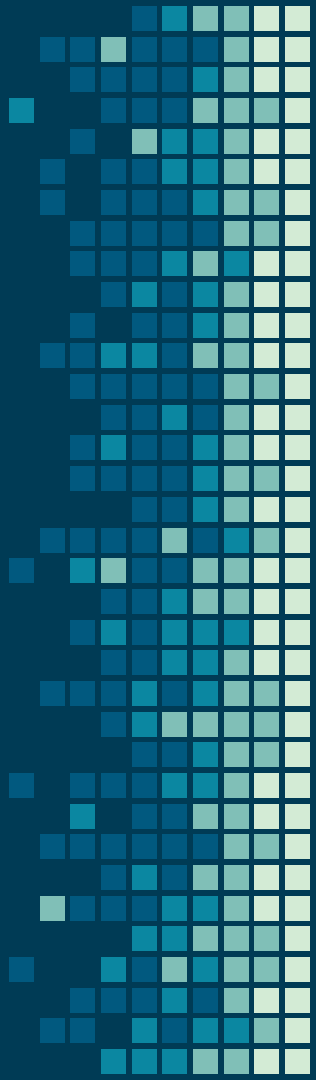
DEFUZZIFICATION ALGORITHM

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:

$$\bar{\theta} = \frac{\int_{\alpha_{initial}}^{\alpha_{final}} \alpha^2 \theta \, d\alpha}{\int_{\alpha_{initial}}^{\alpha_{final}} d\alpha}$$

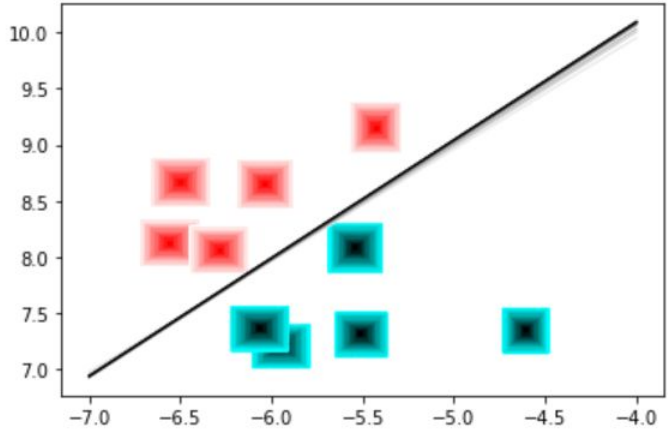
5. After obtaining the defuzzified parameters, we use them to predict on the test set.

EXPERIMENTS ON DATASETS



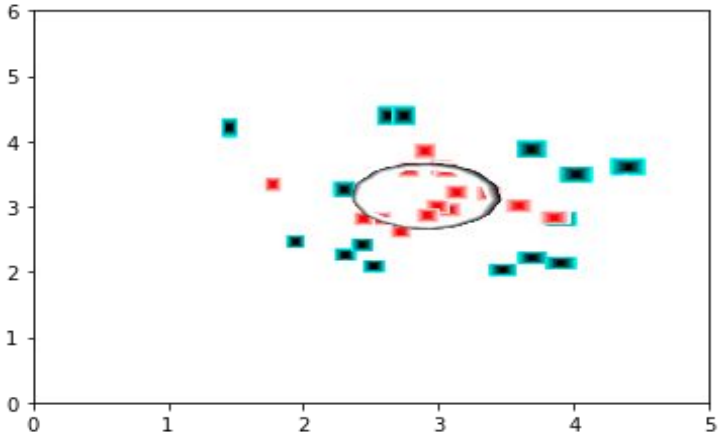
SYNTHETIC DATASET: LINEAR

We have implemented our model in Python language and at first tested it on a linearly separable synthetic dataset generated by the *make_blobs* function in the *scikit-learn* library. The figure below shows the fuzzified data points with the decision boundaries for different alpha values, for this linearly separable synthetic dataset.



SYNTHETIC DATASET: NON-LINEAR

We generated a non-linear synthetic dataset consisting of 30 data points, using the *make_moons* function available in the *scikit-learn* library, and tried our model on it. The figure below shows the fuzzified data points with the decision boundaries for different alpha values, for this non-linear synthetic dataset.



SYNTHETIC DATASET: NON-LINEAR

To get an idea of our model's performance relative to the well-established Classical OC-SVM model, we compared the classification accuracy of the two on the non-linear synthetic dataset mentioned in the previous slide. We found that our model yielded better results compared to the Classical One-Class SVM model. The comparison is summarised in Table below:

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

REAL WORLD DATASET: 'IRIS'

We also compared our model's performance with that of Classical OC-SVM model on the popular **'Iris Dataset'** and found our model to be much more accurate. The Table shows the classification accuracies of our Fuzzy OC-SVM model and that of the Classical OC-SVM model, when they are trained with the data points corresponding to each species of the Iris dataset.

Species	Fuzzy One-Class SVM (accuracy)	Classical One-Class SVM (accuracy)
0	**0.84**	**0.15332**
1	**0.70**	**0.16667**
2	**0.72**	**0.16667**

CONCLUSION

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.


THANK
YOU !
21