Learning with a Reject Option

Dissertation submitted in partial fulfillment of the requirements for the degree of

Master of Technology

in

Computer Science

by

Rajarshi Bhattacharjee

[Roll No: CS1508]

under the guidance of

Prof. Nikhil R. Pal

Electronics and Communication Sciences Unit

Indian Statistical Institute

Kolkata-700108, India

July 2017

M.Tech(CS) DISSERTATION THESIS COMPLETION CERTIFICATE

Student: Rajarshi Bhattacharjee (CS1508)

Topic: Learning with a Reject Option

Supervisor: Prof. Nikhil R. Pal

This is to certify that the thesis titled "***Learning with a Reject Option***" submitted by **Rajarshi Bhattacharjee** in partial fulfillment for the award of the degree of Master of Technology is a bonafide record of work carried out by him under my supervision. The thesis has fulfilled all the requirements as per the regulations of this Institute and, in my opinion, has reached the standard needed for submission. The results contained in this thesis have not been submitted to any other university for the award of any degree or diploma.

Date: Prof. Nikhil R. Pal

Acknowledgements

I would like to thank my supervisor Prof. Nikhil R. Pal for introducing me to this research problem and for his continued guidance and support throughout the year.

Abstract

A major assumption traditional machine learning algorithms make is that the classes encountered during testing phase is always a subset of the classes encountered during training phase. However, in real world applications like biometric recognition, this assumption is violated most of the time. There might be some patterns in the test data that are located far from the training data used to train the classifier. In this scenario, instead of classifying the pattern into any of the *known* classes, the best option will be to reject it. Thus, when that is appropriate, our algorithm needs to have a mechanism to reject patterns instead of classifying them into any of the *known* classes.

In this thesis, we propose two algorithms with a reject option. The first algorithm is an unsupervised one which uses a Self Organizing Map (SOM). SOMs are known to preserve topological properties of the input data like neighbourhood distances and density. We set a rejection threshold based on distances of points mapped to a SOM node. The second algorithm is a two stage rejection algorithm based on Extreme Value Theory. In the first stage, we model the data using a Gaussian Mixture Model for every class and set a rejection threshold based on extreme value distribution of the Mahalanobis distance of the points from the mixture components. In the second stage, we train Support Vector Machines (SVMs) for all the classes in a one-vs-all fashion. We set a rejection threshold based on the extreme value distribution of the SVM scores. To test our algorithms, we simulate an *open set* scenario, where our model is trained using only a subset of classes present in the

dataset. Thus, while testing, there are data from *known* classes which our algorithm should classify and data from *unknown* classes which it should reject. We also analyze our algorithms by discussing their pros and cons and also provide some ideas to improve their performance further.

Contents

Chapter 1

Introduction

Machine learning algorithms generally assume that the classes encountered during testing phase have been already encountered during training phase. In other words, we assume the classes in the test data are a subset of the classes in the training data. However, this assumption is violated most of the time. We may encounter some patterns in the testing phase which cannot be accurately described by any of the classes we have encountered during the testing phase. In this case, the best thing to do will be to not classify the points into any of the previously known classes. Moreover, a test point may come from somewhere far from the training data that were used to design the system. In this case too, the classifier should not make any decision. Thus our algorithm needs a mechanism to reject these points instead of classifying them into any of the known classes.

A simple example to demonstrate the usefulness of having a reject option is a recognition problem. Here, given the input image, we are supposed to classify it into one the classes. In [22], the authors assert that classes in a recognition problem could belong to three basic categories:

- *known classes*: classes labeled with positive labels (that is, labels are distinct and match the corresponding class) encountered during training.

- *known unknown classes*: labeled negative examples that do not belong to any of the known meaningful categories.

- *unknown unknown classes*: classes which haven't been encountered at all.

The authors [22] define an ***open set*** scenario for recognition problems where along with data from *known* classes and *known unknown* classes in the training set, we may encounter data from unknown unknown classes during testing. The data from *known unknown* classes may be treated as an explicit "other" class while designing the classifier. Learning with a reject option enables us to reject data belonging to *unknown unknown classes* and thus, effectively it solves the open set problem.

Having a reject option is very helpful in a number of situations in real life scenarios. One example is the problem of biometric recognition. In high security installations, we may never want a misindentification to occur. The reject option will also drastically improve the classification accuracy in common recognition applications like Optical Character Recognition (OCR) and photo and video tagging without constraints on the input space.

In this thesis we propose some new algorithms to tackle the open set problem by learning with a reject option. Our objective is to classify the data correctly if it belongs to one of the "known" classes while rejecting it otherwise. The rest of this dissertation is organised as follows. In Chapter 2, we describe Extreme Value Theory. In Chapter 3, we review some previous works done on this topic. We introduce our Self Organizing Map based reject algorithm in Chapter 4. In Chapter 5, we explain another algorithm with a reject option based on Extreme Value Theory. Finally, in Chapter 6 we present our experimental results with real data and end with some concluding remarks in Chapter 7.

Chapter 2

Preliminary Knowledge

In this chapter, we describe some fundamentals of Extreme Value Theory (EVT) which will be useful in subsequent chapters including the survey of literature.

2.1 Extreme Value Theory

Extreme value theory is a branch of statistics that deals with modeling extreme deviations from the median value of probability distributions. It does this by trying to analyze the tails of probability distributions because 'extreme values' generally belong to the tails. A common example of the use of EVT can be found in civil engineering where one may need to build structures which are resistant to a 100-year-flood. By definition, a 100-year-flood occurs once in 100 years (has a 1% chance to occur in any given year) and so relevant data to model these 'extreme' events may not be available. So, it is needed to appropriately extrapolate these events from the normal flood data. That is, we want to model the tail of the distribution using data from around the median. Another example can be found in finance where EVT is used to model rare events like stock market crashes or for measuring financial risks [11].

Extreme value theory is practically applied using two approaches:

- **Block Maxima Approach**: In this approach, the maxima or minima of a block of data is sampled repeatedly to get a series of maxima or minima. Then, the distribution of these extreme values is modeled using one of the 'Extreme Value Distributions' [9].

- **Peaks over Threshold**: In this approach, the exceedances of values above a fixed threshold is modeled using the Generalized Pareto distribution [15].

In this dissertation, we apply Extreme Value Theory in a similar fashion as done by the authors of [23] and [22]. The following theorem is known as the Extreme Value Theorem (also known as the Fisher-Tippet theorem) [13] which states the following:

Theorem 1 *Let* $(s_1, s_2, s_3, ...)$ *be a sequence of independent and identically distributed samples. Let* $M_n = max\{s_1, ..., s_n\}$*. If a sequence of pairs of real numbers* (a_n, b_n) *exists such that each* $a_n > 0$ *and*

$$\lim_{n\to\infty} P\left(\frac{M_n - b_n}{a_n} \leq x\right) = F(x) \tag{2.1}$$

where F is a nondegenerate distribution function,then F belongs to one of three extreme value distributions:

1. *Gumbel distribution*

2. *Fretchet distribution*

3. *Weibull distribution*

Thus, from the above theorem, it is clear that if a limiting distribution of the maxima of independent and identically distributed samples exist, it is bound to follow one of the three extreme value distributions. These distributions, as mentioned, are the Gumbel or type I, Fretchet or type II and Weibull or type III extreme value distribution. The three types of distributions can be unified into a single distribution called the Generalized Extreme Value (GEV) distribution [23], the probability

density function of which is given by:

$$GEV(t) = \begin{cases} \frac{1}{\lambda} e^{-v^{-1/k}} v^{-(\frac{1}{k}+1)}, & k \neq 0, \\ \frac{1}{\lambda} e^{-(x+e^{-x})}, & k = 0, \end{cases} \tag{2.2}$$

where $x = \frac{t-\tau}{\lambda}$, $v = (1 + k\frac{t-\tau}{\lambda})$, where k, λ and τ are shape, scale and location parameters respectively. The three cases $k = 0, k > 0$, and $k < 0$ correspond to the Gumbel, Fretchet and Reversed Weibull distributions respectively. The Reversed Weibull distribution is nothing but the weibull distribution reflected along the Y-axis and is defined on $(-\infty, 0]$ as opposed to the weibull which is defined on $[0, \infty)$. The Weibull and Reversed Weibull distributions are used in cases the data are lower or upper bounded respectively. Gumbel and Fretchet distibutions can be used when the data are unbounded. We will be using the Gumbel and Weibull distributions later on.

Chapter 3

Related Work

In this chapter we provide a review of some of the works done previously on this topic.

3.1 Related Work

Probabilistically modeling the data to set a rejection threshold on the probability is a widely used method to set a reject option. One of the earliest and most well known works' where a reject option was first proposed, was by Chow [5]. Chow proved that if we have the prior probability and class conditional densities of every class, then the optimum rejection rule is always a threshold on the maximum posterior probability. However, it is assumed that there is only one reject class and hence, the results of this paper might be difficult to extend to the open set problem. Further reject options based on probabilities were introduced by Dubuisson and Mason [7] and Muzzolini et al. [17].

Support Vector Machines (SVM) with kernels, when trained with two classes try to find the optimal hyperplane separating the classes in the feature space defined by the kernel. One-class support vector machine is used when data from only one class are available and for a test point the goal is to find if it belongs to that class or not.

One-class SVM was introduced by Scholkopf et al. [25]. It basically tries to find a hyperplane to separate the data from the origin in the feature space by maximizing the distance from this hyperplane to the origin. However, one must specify an upper bound on the fraction of outliers in the dataset beforehand. There is another type of one-class SVM called Support Vector Data Description (SVDD), introduced by Tax and Duin [26]. The SVDD tries to find a minimum volume hypersphere in the feature space enclosing the data. The resulting hypersphere is characterized by its centre and radius which is found as a solution to an optimization problem. However, the decision boundary found by it is often found to contain too much open space along with the data inside. Thus, one-class SVMs often do not perform well in practice.

Apart from one-class SVMs, binary SVMs have also been suitably modified to incorporate a reject option. Scheirer et al [24] introduced the "1 vs Set Machine" which uses two hyperplanes- a near plane and a far plane instead of one to effectively model the class decision boundary. However, this works with linear kernels and has so far, to our knowledge, not been generalized for non linear kernels. Bartlett and Wegkamp [1] introduced a modified loss function for support vector machines instead of the hinge loss. This new loss function basically tries to make classifying outliers more costly than normal data.

Neural networks (MLPs) with reject option have also been developed. Chakraborty and Pal [3] developed a scheme to generate points outside the boundary of a class. Using this, for each class, they trained a multi-layer perceptron with the class data and points outside that class's boundary. Finally, they combined them together so that the combined network has better classification ability. It is also able to distinguish points which lie outside class boundaries better than a normal multi-layer perceptron. This scheme also equipped an MLP with incremental learning ability. However, generating points for high dimensional data using this method is costly.

Unsupervised and semi-supervised approaches to solve the open set problem are especially useful when there is no class label information. One of the algorithms

we propose in this thesis is based on the Self Organizing Map (SOM). The SOM is an unsupervised algorithm to find representative prototypes of the data while also preserving the topology of the data (the map preserves distance relationships between neighbouring points in the dataset). Though it is mostly used for clustering or visualization of the data, it has also been used previously for novelty detection. Labib and Vemuri [14] use a SOM to classify ethernet network data in real time and also detect network intrusions. They train a SOM on normal network data and graphically visualize the data in two dimensions. By visually inspecting the clusters formed around the SOM nodes, they try to detect network intrusions which are projected far away from the normal clusters. Similar work was done by Ramadas et al. [19] where they trained a SOM to detect network anomalies by inspecting the distance of data points from the best matching unit in the SOM.

Next, we look at prior work using Extreme Value Theory (EVT) in novelty detection. Roberts [20] used EVT to model the extreme values of a Gaussian Mixture Model. Some improvements to the method were proposed more recently by Clifton et al. [6]. We will be looking at these methods in detail later when we describe our second algorithm. More recently, EVT has been used on the output of deep neural networks to reject points [2]. In [2] instead of using a softmax layer, an "open max" layer is used to turn the outputs from the penultimate layer of the neural network into probabilities using EVT.

3.2 Open Set Problem

In this section, we provide a detailed review of one of the recent works on the **Open Set** problem by Scheirer et al. [22] since we will be comparing our algorithms with the algorithms proposed in this paper. In this paper, an algorithm with two stage rejection procedure called WSVM (Weibull calibrated Support Vector Machine) is proposed.

Let our dataset be $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ...(\boldsymbol{x}_n, y_n)\}$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ are d-dimensional features and y_i are the class labels $\forall i \in \{1, 2, ..., n\}$. The number of unique class labels in the data set is c and they are represented by $\{l_1, l_2, ...l_c\}$. Thus $y_i = l_j$ indicates that the ith data point belongs to class j where $j \in \{1, 2, ...c\}$.

Next, consider all data points $(\boldsymbol{x}_i, y_i)$ where $y_i = l_j$ i.e. all points belonging to the jth class. The authors train a one-class SVM with RBF kernel with this data. They then calculate the SVM decision scores for all the points. The hyperplane divides the whole space into a positive space containing the training data from the jth class and a negative space containing the origin. This is because the points classified as belonging to the class j get positive scores while others get negative scores. The absolute value of a score indicates distance from the hyperplane. The corresponding hyperplane serves as a class decision boundary for the first class. The points closest to the hyperplane are most likely to be misclassified. Hence, the smallest q_j^o positive SVM scores are used to fit an extreme value distribution on them where tail size $q_j^o = 1.5\times$(No of support vectors for positive class). Since the scores are bounded below, they fit a Weibull distribution using the scores. Let the smallest positive score be s_j^o. The location parameter is then fixed as

$$\nu_{o,j} = s_j^o \tag{3.1}$$

Then the shape and scale parameters $\lambda_{o,j}$ and $\kappa_{o,j}$ are estimated using the maximum likelihood estimation.

Then, they train a binary SVM in one vs all manner for the jth class. That is, if the ith point has $y_i = l_j$, it is labeled as +1 and rest of the points as -1 and then a binary SVM is trained. Here also the points closest to the decision boundary are most likely to be misclassified. So, they considered the smallest q_j^+ positive SVM scores for the positive class and the largest q_j^- negative SVM scores. For the positive scores, they fit a Weibull distribution as they are bounded from below. For the negative scores which are bounded from above, a Weibull distribution is fit after negating the scores. They call data with positive scores as match data and data with

negative scores as non-match data respectively. For the match data, they again fix the Weibull location parameter $\nu_{\eta,j}$ as the smallest positive match score. Then $\lambda_{\eta,j}$ and $\kappa_{\eta,j}$, the Weibull scale and shape parameters are estimated from the smallest q_j^+ positive scores using maximum likelihood estimation. Similarly, they estimate the Weibull parameters $\nu_{\psi,j}$, $\lambda_{\psi,j}$ and $\kappa_{\psi,j}$ for non match data using the largest q_j^- scores. These Weibull parameter estimates are found for all the classes (i.e. $\forall j \in \{1, 2, ..c\}$) using the training data. The algorithm is described in Algorithm 1.

Finally, given a new sample $\boldsymbol{x}$, the decision is made as described in Algorithm 2. For the ith class, let the one-class SVM score function be $f_i^o(\boldsymbol{x})$ and let the binary SVM score be $f_i(\boldsymbol{x})$. Then, the probability of not belonging to the ith class is given by the cumulative distribution function (CDF) of the Weibull distribution from the positive one class SVM scores. Thus, the probability of belonging to the ith class is just 1-(Weibull CDF of $\boldsymbol{x}$). This is given by:

$$P_{o,i}(\boldsymbol{x}) = e^{-\left(\frac{|f_i^o(\boldsymbol{x}) - \nu_{o,j}|}{\lambda_{o,j}}\right)^{\kappa_{o,j}}} \tag{3.2}$$

If $P_{o,i}(\boldsymbol{x}) > \delta_T$ where δ_T is some predetermined threshold, then we calculate another probability using the binary SVM score. If $P_{o,i}(\boldsymbol{x}) < \delta_T$ the probability score of $\boldsymbol{x}$ belonging to class i is 0. The probability of not belonging to the ith class is again given by the cumulative distribution function (CDF) of the Weibull distribution from the match data. So, like the one class SVM, the probability of belonging to the ith class is given by:

$$P_{\eta,i}(\boldsymbol{x}) = e^{-\left(\frac{|f_i(\boldsymbol{x}) - \nu_{\eta,j}|}{\lambda_{\eta,j}}\right)^{\kappa_{\eta,j}}} \tag{3.3}$$

Next, from the non-match data, one can calculate the probability of not belonging to the non-match data. This is given by the CDF of the non-match Weibull which is:

$$P_{\psi,i}(\boldsymbol{x}) = 1 - e^{-\left(\frac{|-f_i(\boldsymbol{x}) - \nu_{\psi,j}|}{\lambda_{\psi,j}}\right)^{\kappa_{\psi,j}}} \tag{3.4}$$

Note that we must negate the SVM score to $-f_i(\boldsymbol{x})$ as we had calculated the Weibull parameters by negating the non-match scores. The final WSVM score of $\boldsymbol{x}$ belonging

to class i is calculated as :

$$T_i(\boldsymbol{x}) = P_{\eta,i}(\boldsymbol{x}) \times P_{\psi,i}(\boldsymbol{x}) \tag{3.5}$$

This score can be interpreted as the "the probability that the input is from the positive class (i.e ith class) AND NOT from any of the known negative classes" [22]. Finally, the label y is assigned to $\boldsymbol{x}$ as:

$$\begin{gathered} y = \underset{l_i \;\; i\in\{1,2,..,c\}}{\operatorname{argmax}} \; T_i(\boldsymbol{x}) \times t_i \\ \text{subject to } T_i(\boldsymbol{x}) \times t_i \geq \delta_R \end{gathered} \tag{3.6}$$

where the δ_R is a second threshold and t_i is an indicator variable for the ith class which is 1 if $P_{o,i}(\boldsymbol{x}) \geq \delta_T$ and 0 otherwise. If the condition $T_i(\boldsymbol{x}) \times t_i \geq \delta_R$ is not satisfied for any class, $\boldsymbol{x}$ is rejected. Thus, by setting the two parameters δ_T and δ_R, we get a classifier which can also reject inputs.

The value δ_R is set according to the **Openness** of the problem which measures the proportion of unknown classes in the test data. The higher the number of unknown classes in the test data, the higher the openness of the problem and hence, the higher should be the value of the threshold. This is because intuitively, higher number of unknown classes means the probability of the input being from an unknown class is higher. So, one should reject with lower confidence and hence choose a higher threshold. In [22], openness is defined as

$$Openness = 1 - \sqrt{\frac{2 \text{ x } |training\ classes|}{|testing\ classes| + |target\ classes|}} \tag{3.7}$$

Here, $|training\ classes|$ and $|testing\ classes|$ are the number of classes used in training and testing the model respectively and $|target\ classes|$ is the total number of classes to be identified by the model. We set the value of δ_R as $0.5 \times Openness$. We note here that for any real application, the number of unknown classes (and thus, the value of $|target\ classes|$) is not known before hand. Hence, it might be difficult to calculate openness for real world data.

Algorithm 1: WSVM Model Fitting

Data: $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ...(\boldsymbol{x}_n, y_n)\}$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{l_1, l_2, ...l_c\}$

Input:

Pre-trained one-class SVM for each class, with SVM score functions $f_i^o()$ and support vectors α_i^o for the ith class $\forall i \in \{1, 2, ..., c\}$;

Pre-trained 1-vs-All binary SVM for each class, with SVM score functions $f_i()$ and support vectors α_i^+, α_i^- for match and non-match data respectively for the ith class $\forall i \in \{1, 2, ..., c\}$;

Tail size multiplier $\theta = 1.5$;

for *i=1 to c* **do**

  Let the set of binary svm scores be $S_i = \{f_i(\boldsymbol{x}_j)\}$ and the set of one-class svm scores be $S_i^o = \{f_i^o(\boldsymbol{x}_j)\}$ $(\forall j \in \{1, 2, ..., n\}$ with $y_j = l_i)$;

  Let $q_i^+ = \theta \times |\alpha_i^+|$, $q_i^- = \theta \times |\alpha_i^-|$ and $q_i^o = \theta \times |\alpha_i^o|$;

  Let d_i^o be the smallest q_i^o positive values from S_i^o ;

  Let d_i^+ be the smallest q_i^+ positive (match) scores from S_i and d_i^- be the largest q_i^- negative (non-match) scores from S_i ;

  $[\nu_{o,i}, \lambda_{o,i}, \kappa_{o,i}]$=Estimates of Weibull parameters from d_i^o ;

  $[\nu_{\eta,i}, \lambda_{\eta,i}, \kappa_{\eta,i}]$=Estimates of Weibull parameters from d_i^+ ;

  $[\nu_{\psi,i}, \lambda_{\psi,i}, \kappa_{\psi,i}]$=Estimates of Weibull parameters from $-d_i^-$(negate scores to make them positive) ;

end

Output: W_i=$[\nu_{o,i}, \lambda_{o,i}, \kappa_{o,i}, \nu_{\eta,i}, \lambda_{\eta,i}, \kappa_{\eta,i}, \nu_{\psi,i}, \lambda_{\psi,i}, \kappa_{\psi,i}]$ $\forall i \in \{1, 2, ..., c\}$

Algorithm 2: WSVM Label Estimation

Input:

Input data point $\boldsymbol{x}$;

Pre-trained one-class SVM for each class $i \in \{1, 2, ..., c\}$;

Pre-trained 1-vs-All binary SVM for each class $i \in \{1, 2, ..., c\}$;

Parameter estimates W_i for each class $i \in \{1, 2, ..., c\}$;

Set δ_T and $\delta_R = 0.5 \times Openness$;

for *i=1 to c* **do**

  Calculate $P_{o,i}(\boldsymbol{x})$ using Equation 3.2. ;

  Calculate $P_{\eta,i}(\boldsymbol{x})$ using Equation 3.3. ;

  Calculate $P_{\psi,i}(\boldsymbol{x})$ using Equation 3.4. ;

  Set t_i=0 ;

  if $P_{o,i}(\boldsymbol{x}) \geq \delta_T$ **then**

    Set t_i=1 ;

  end

  Calculate $T_i(\boldsymbol{x})$ using Equation 3.5 ;

end

Find label y using 3.6 or $y = 0$ if $T_i(\boldsymbol{x}) \times t_i < \delta_R \; \forall i \in \{1, 2, ..., c\}$

Output: Label y of $\boldsymbol{x}$ if accepted else reject $\boldsymbol{x}$

Chapter 4

Classifier with Reject Option using Self Organizing Maps

4.1 Kohonen's Self-Organizing Map

Kohonen's Self-Organizng Map (SOM) is a type of artificial neural network which was introduced by Teuvo Kohonen in the early 1980's. SOM's learn by a technique called *Competitive Learning*. Neurons/nodes compete among themselves to be activated or fired. Finally, the neuron with the highest output, which is the most activated neuron (also called the "winning neuron") fires. This is supposed to be biologically motivated from the neuron structure and activation in the brain. We look at the architecture and learning algorithm of SOM in detail next.

4.1.1 SOM Architecture

SOM has two layers consisting of an input layer and an output layer. The number of nodes of the input layer is the same as the dimension d of the input data. Each node in the input layer is connected to all the nodes in the output layer. These connections have weights associated with them which we need to learn from the

input data. Thus, each output node is associated with a d-dimensional weight vector $\boldsymbol{w}$. The output nodes are arranged in a p-dimensional regular map or grid. Common choices are one-dimensional or two-dimensional (rectangular) map. We will be using a one-dimensional (1-D) map as it is more flexible in placement of prototypes and it has given better results experimentally for our problem. In a 1-D SOM, every node has two neighbours (left and right) except the first and last node which have one neighbour each. Let $r_{i,j}$ denote the lateral distance between ith node and jth node of the output layer. In case of 1-D SOM, it is just the absolute value of the difference between node positions:

$$r_{i,j} = |i - j| \tag{4.1}$$

Suppose there are m SOM nodes in the output layer. Let input vectors be $\boldsymbol{x} \in \mathbb{R}^d$. The weights are represented by $\{w_{i,j}\}$ where $i \in \{1, 2, ..., m\}$ and $j \in \{1, 2, ..d\}$. So, $w_{i,j}$ implies the weight is of the connection from jth input node to ith output node. There are a total of $m \times d$ weights. Weight vector for the ith output node is $\boldsymbol{w}_i = (w_{i,1}, w_{i,2}, ...w_{i,d})^T$.

4.1.2 SOM Training Algorithm

Let the input data be $\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_1, ...\boldsymbol{x}_n\} \subset \mathbb{R}^d$. We initialize the weights $\{w_{i,j}\}$ with random values between 0 and 1. Next, suppose at the iteration t, the input $\boldsymbol{x}$ is given to the SOM. We first find the node with the minimum euclidean distance (maximum similarity) to $\boldsymbol{x}$. We call this the "winning" node. That is, the winning node on presenting input $\boldsymbol{x}$ is

$$\boldsymbol{w}_k = \underset{i}{\operatorname{argmin}} ||\boldsymbol{x} - \boldsymbol{w}_i||, \quad i \in \{1, 2, ...n\} \tag{4.2}$$

where $||\quad||$ represents the Euclidean distance. Now, we need to update the weight of the winning node as well as its neighbors. We update the ith weight as follows:

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta(t) h_{i,k}(t)(\boldsymbol{x} - \boldsymbol{w}_i(t)) \tag{4.3}$$

where $\boldsymbol{w}_i(t)$ represents weight vector associated with the ith node at iteration t, $\eta(t)$ is the learning rate at iteration t and $h_{i,k}(t)$ is the value of a topological neighborhood function at iteration t with respect to the winning node $\boldsymbol{w}_k$. The choice of $\eta(t)$ and $h_{i,k}(t)$ is explained next. In SOM, learning happens in two phases [12]:

- *Self-organizing or ordering phase*: This is the initial phase during which the topological ordering of nodes take place. During this phase, $\eta(t)$ begins with a fairly large value and gradually decreases. However, it shouldn't decrease too much (usually not below 0.01). So, during this phase

$$\eta(t) = \eta_o \exp\left(-\frac{t}{\tau_2}\right) \tag{4.4}$$

  where η_o is the initial learning coefficient and τ_2 is the number of steps to be used in the ordering phase. In our experiments,we have used $\eta(0) = \eta_o = 0.1$ and $\tau_2 = 1000$. Also, during this phase, we begin with a very large neighborhood and decrease it smoothly to 1, that is, till the winning node is the only node being updated. As usually done, we set it as:

$$h_{i,k}(t) = \exp\left(-\frac{r_{i,k}^2}{2\sigma^2(t)}\right) \tag{4.5}$$

  where $\sigma(t)$ is the width of the neighborhood function given by

$$\sigma(t) = \sigma_o \exp\left(-\frac{t}{\tau_1}\right) \tag{4.6}$$

  σ_o is equal to the "diameter" of the lattice given by $(m-1)$ for the 1D SOM, $r_{i,k} = |i-k|$ and $\tau_1 = \frac{1000}{\log \sigma_o}$.

- *Convergence phase*: This phase is needed to fine tune the map and to provide an accurate representation of the input space. The number of iterations in this phase is almost 500 times the number of nodes in the output layer. In this phase, we fix $\eta(t)$ at 0.01 for all iterations and

$$\begin{aligned} h_{i,k}(t) &= 1 \quad if \quad i = k \\ &= 0 \quad otherwise \end{aligned} \tag{4.7}$$

  Thus, only the winning node is updated in this phase.

The SOM learning algorithm is described in Algorithm 3.

4.2 Designing the SOM Based Classifier

SOM's are known to preserve the topological properties of the input space like neighbourhood distances and density of the input data. This makes SOMs very effective to implement a local neighborhood distance based reject classifier. Now, we describe the algorithm to implement a distance based reject option using SOM. We first train a SOM with the training data using the algorithm described in Algorithm 3. Then, we give the SOM inputs from the training set one by one. For every node, we store the maximum distance from that node to all the points for which it is the winner, i.e., the maximum distance from the node to all the points in the training set which have that node as its winning node (D). We also store the mean and the standard deviation of these distances for every node. For the ith node the maximum distance to it is denoted by D_i and the mean and the standard deviation of distances by μ_i and σ_i respectively. This algorithm is explained in Algorithm 4. Now, when a new input arrives, we first find the winning node (say the ith node) and its euclidean distance from the winning node (say d_i). Now we can decide whether to reject the input using two different criteria. For the first method, we reject this input if this distance (d_i) is greater than D_i. This is described in Algorithm 5. The maximum distance from the node may be too sensitive a criteria in some cases (for example, when there are outliers). Thus in the second method, we reject the input, if d_i is greater than $\mu_i + \sigma_i$. This is described in Algorithm 6. We call these algorithms **SOM_Reject1** and **SOM_Reject2** respectively. We shall comment on these two versions in later sections, while discussing the experimental results.

Algorithm 3: SOM Training

Input:

$\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ...\boldsymbol{x}_n\} \subset \mathbb{R}^d$;

MaxStep1 ; `// steps in Ordering phase`

MaxStep2 ; `// steps in Convergence phase`

m ; `// No of SOM nodes`

$t = 1$;

Initialize $\{w_{i,j}\}$ with random numbers between 0 and 1 $\forall i \in \{1, 2, ..., m\}$ and $\forall j \in \{1, 2, ...d\}$

`// Ordering phase`

while $t \leq$*MaxStep1* **do**

  $i = modulo(t, n)$; `// find remainder on dividing t by n`

  $\boldsymbol{x} = \boldsymbol{x}_i$; `// select ith input`

  Find winning node k using 4.2 ;

  Update $\boldsymbol{w}_i$ as $\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta(t)h_{i,k}(t)(\boldsymbol{x} - \boldsymbol{w}_i(t))$where $\eta(t)$ is updated using Equation 4.4 and $h_{i,k}(t)$ using Equation 4.5 $\forall i \in \{1, 2, ..., m\}$;

  t=t+1 ;

end

`// Convergence phase`

$t = 1$;

while $t \leq$*MaxStep2* **do**

  $i = modulo(t, n)$; `// find remainder on dividing t by n`

  $\boldsymbol{x} = \boldsymbol{x}_i$; `// select ith input`

  Find k using 4.2 ;

  $\boldsymbol{w}_k(t+1) = \boldsymbol{w}_{k)}(t) + 0.01(\boldsymbol{x} - \boldsymbol{w}_k(t))$;

  t=t+1 ;

end

Output: $\boldsymbol{w}_i \ \forall i \in \{1, 2, ...m\}$

Algorithm 4: Find Max Distances from SOM Nodes

Input:

$\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ...\boldsymbol{x}_n\} \subset \mathbb{R}^d$;

Trained 1D SOM weights $\boldsymbol{w}_i$ $\forall i \in \{1, 2, ..., m\}$ using data $\mathbf{X}$ and Algorithm 3

Initialize $D_i = 0$ $\forall i \in \{1, 2, ..., m\}$;

for *i=1 to n* **do**

  $\boldsymbol{x} = \boldsymbol{x}_i$;

  Find winning node k using 4.2;

  Let $dist = ||\boldsymbol{x} - \boldsymbol{w}_k||$;

  if $dist > D_k$ **then**

    $D_k = dist$

  end

end

Calculate mean μ_i and standard deviation σ_i of distances from for each node.

Output: D_i,μ_i and σ_i $\forall i \in \{1, 2, ..., m\}$

Algorithm 5: SOM_Reject1

Input:

Input data point $\boldsymbol{x}$;

Trained 1D SOM weights $\boldsymbol{w}_i$ using training data $\mathbf{X}$ and Algorithm 3 and D_i from Algorithm 4 $\forall i \in \{1, 2, ..., m\}$

Find winning node k using Equation 4.2;

Let $dist = ||\boldsymbol{x} - \boldsymbol{w}_k||$;

if $dist > D_k$ **then**

  Reject $\boldsymbol{x}$;

end

Algorithm 6: SOM_Reject2

Input:

Input data point $\boldsymbol{x}$;

Trained 1D SOM weights $\boldsymbol{w}_i$ using training data $\mathbf{X}$ and Algorithm 3 and D_i from Algorithm 4 $\forall i \in \{1, 2, ..., m\}$

Find winning node k using Equation 4.2;

Let $dist = ||\boldsymbol{x} - \boldsymbol{w}_k||$;

if $dist > \mu_k + \sigma_k$ **then**

    Reject $\boldsymbol{x}$;

end

4.3 Results and Visualizations with SOM

We now look at the results of testing the SOM algorithm with some simple datasets and their corresponding visualizations.

4.3.1 Synthetic Dataset 1

We have designed a synthetic dataset to show the effectiveness of our algorithm. The dataset consists of 804 two dimensional points. Out of these, 800 are arranged in 6 clusters while 4 are outliers. The points for testing are generated at equally spaced intervals of 0.05 along the X-axis and Y-axis in the range $[-1, 8] \times [-1, 8]$. We train two SOMs with 80 and 50 nodes respectively on this data and reject points using Algorithm 5 (Maximum distance criteria). The results along with the training and test sets and SOM nodes are shown in Figures 4.2 and 4.3. The green points are the training points. The red points are SOM nodes. The blue points are the accepted points while the black points are rejected points.

We see the SOM with 80 nodes performs very well in capturing the structure of the data and in deciding which points to reject. We specifically notice that the four

outliers do not have too much effect on the rejection ability of the SOM. A single node is assigned to the outlier at the top right and to the two outliers in the middle. Thus, the D_i (maximum distanced from nodes) of these nodes is almost zero. Hence, these nodes end up rejecting almost all the points around the outliers. In contrast to this, the SOM with 50 nodes does not assign a unique node to the two outliers in the middle. Thus, the nodes to which they have been assigned end up having a large maximum distance. This results in them accepting points that should have been rejected (denoted by the large blue region in the middle). Thus, it is crucial we have sufficient SOM nodes to get a good representation of the data. Alternatively, instead of maximum, we can use some other criterion that is not much influenced by the maximum distance or position of outliers. We shall explore this later. Another interesting observation is that some SOM nodes in both cases do not have any points assigned to them (for example, the node between the two big clusters at the top). However, they do not affect the rejection ability of the SOM as their D_i's are equal to zero and they reject any point assigned to them. Moreover, any node with zero or less than k points (where k is some threshold) assigned to it could (should) be deleted.

4.3.2 Synthetic Dataset 2

Synthetic dataset 2 consists of 1000 two dimensional points; 500 of these are generated randomly from a Gaussian distribution centered at $(3, 3)^T$ with unit variance along X and Y axes. The remaining 500 points are generated uniformly at random within a circle centered at $(10, 3)^T$ with radius 3. Test points are generated like in the case of Synthetic Dataset 1 within a box $[-2, 14] \times [-2, 7]$ at intervals of 0.05 along X and Y axes. A SOM with 80 nodes is trained. We use the maximum distance rejection criteria (Algorithm 5) here.The result is shown in Figure 4.5. The color scheme is the same as before. Here also the SOM represents the points very nicely and usually rejects what should be rejected. We say usually because for a prototype

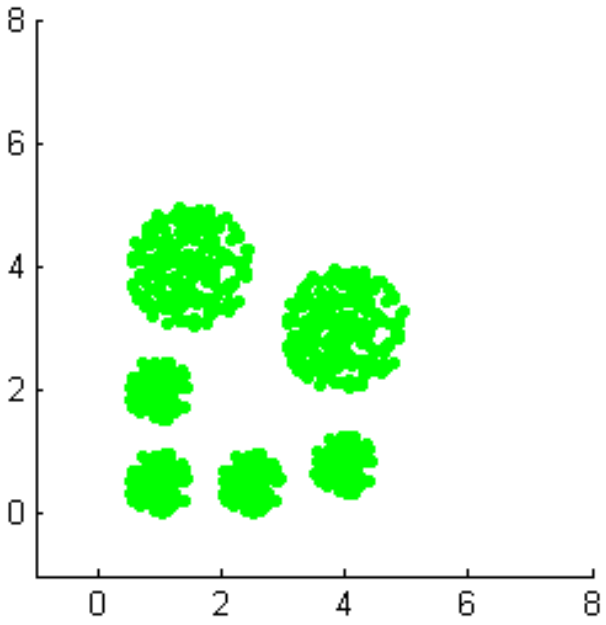

Figure 4.1: Synthetic dataset 1

towards the bottom-left corner, the rejection criterion is more relaxed. We also note that in both clusters, there are some holes where data points are rejected. Although this is reasonable given the dataset, one may argue against it because here we know about the class information.

4.3.3 Synthetic Dataset 3

Synthetic dataset 3 consists of 600 points. 300 of these are randomly generated within a circle centered at $[3, 3]$ with radius 3. The rest 300 are generated in an annular region between circles of radius 4 and 5 centered at $[3, 3]$. Test points have generated within a box $[-4, 10] \times [-4, 10]$ at intervals of 0.05 along X and Y axis. A SOM with 60 nodes is trained and the rejection criteria is maximum distancef rom the SOM nodes. The result is shown in Figure 4.7. The color scheme is same as before. Here also the SOM represents the points very nicely and rejects points well.

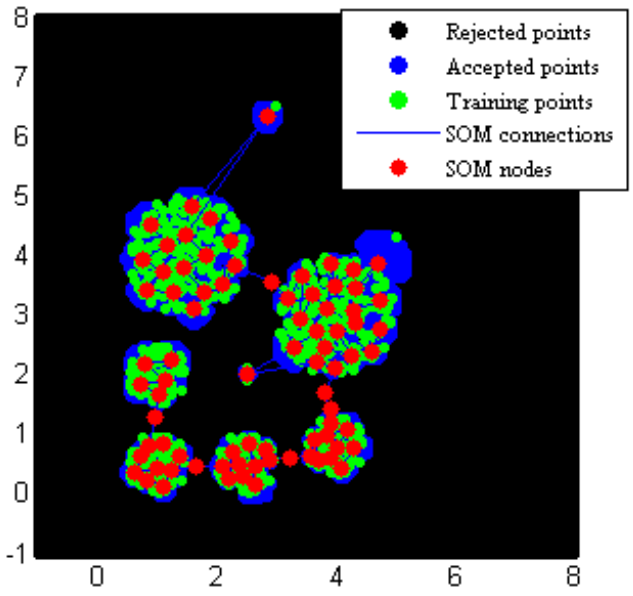

Figure 4.2: SOM_Reject1 with Synthetic dataset 1 with 80 SOM nodes

4.3.4 Iris dataset

The Iris dataset [10] contains 150 points, each with 4 features. The points are from 3 different classes, each containing 50 points. It is used for testing a variety of classifiers. Here, for testing, we leave one class out in turn and train the SOM using remaining two classes. Then, the left out class becomes the test set and we see how many points of the test set the SOM rejects successfully. We use the maximum distance rejection criteria. The number of SOM nodes is fixed at 20. For visualization, we take the all the data points as well as the SOM weights and do Sammon's projection [21] to reduce the number of dimensions to 2. The resulting visualizations are shown in Figures 4.8, 4.9 and 4.10 where respectively class 1, class 2 and class 3 are left out. The SOM nodes are shown in red. The training points are shown in black. The points successfully rejected are shown in blue and the points mistakenly accepted are shown in green. We see that when class 1 is left out, SOM successfully rejects all the points. However, when class 2 is left out, the SOM fails to reject 2 points. Similarly,

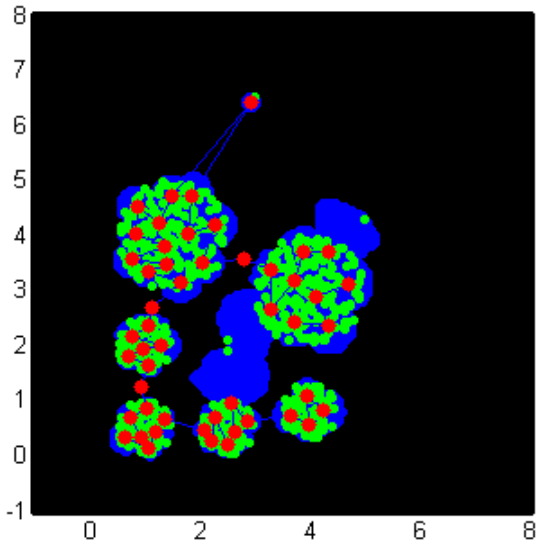

Figure 4.3: SOM_Reject1 with Synthetic dataset 1 with 50 SOM nodes

when class 3 is left out, the SOM fails to reject 6 points. The possible reason for this is the overlap between class 2 and class 3.

To investigate further, we repeat the experiments by training a SOM with 10 nodes separately for each class. The results of leaving out class 1, class 2 and class 3 are shown in Figures 4.11, 4.12, and 4.13 respectively. We find that when class1 is left out, the SOM successfully rejects all the points from class 1 but when class 2 is left out, the SOM mistakenly accepts 3 points. When class 3 is left out, it accepts 6 points that should have been rejected. So, the SOM performance does not improve when training the SOM separately with single classes.

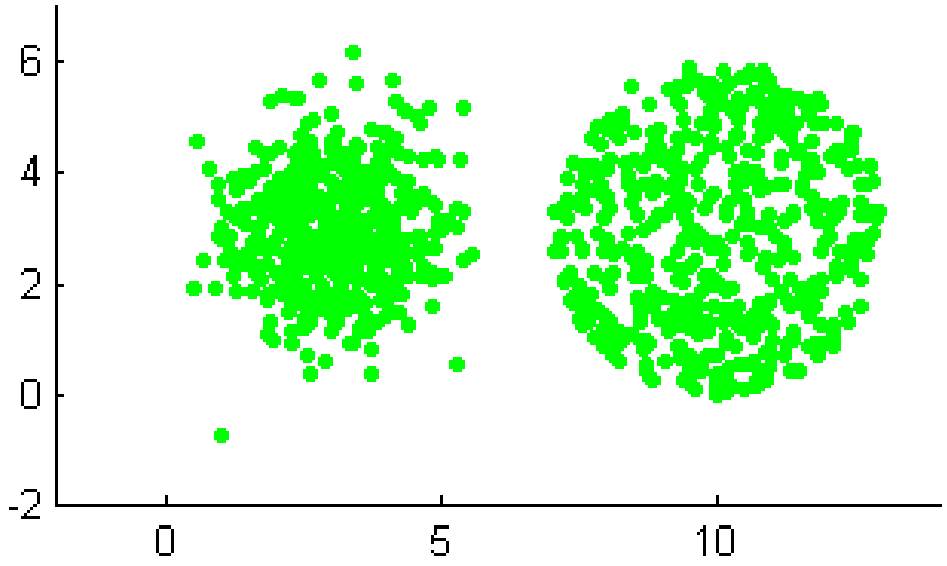

Figure 4.4: Synthetic dataset 2

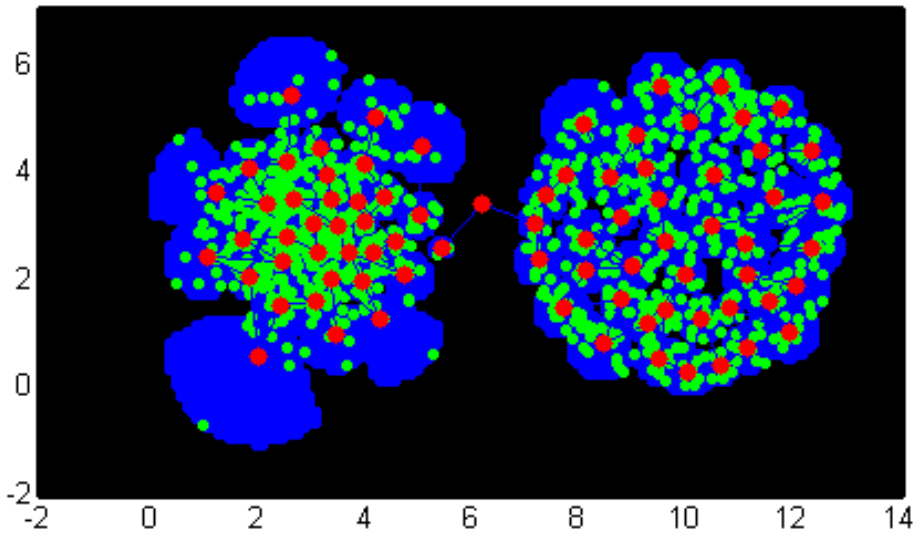

Figure 4.5: SOM_Reject1 with Synthetic dataset 2

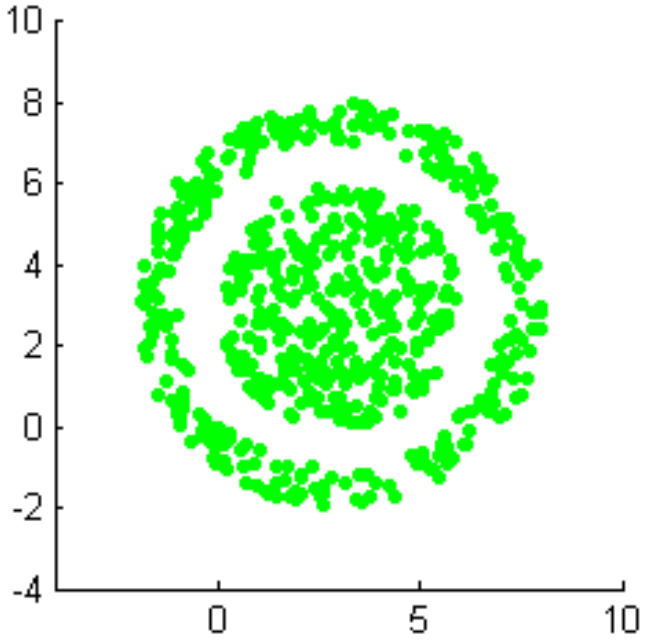

Figure 4.6: Synthetic dataset 3

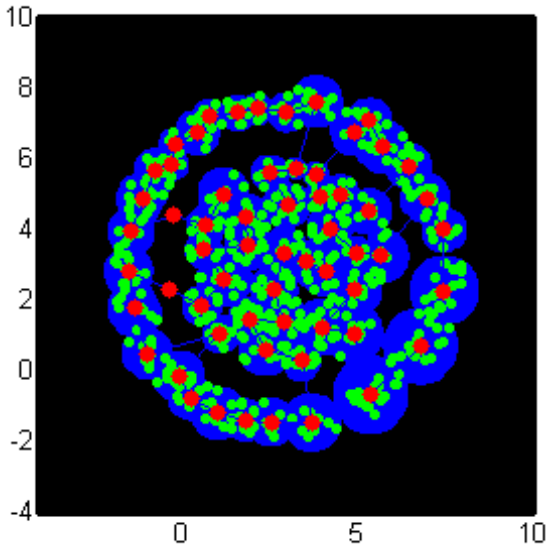

Figure 4.7: SOM_Reject1 with Synthetic dataset 3

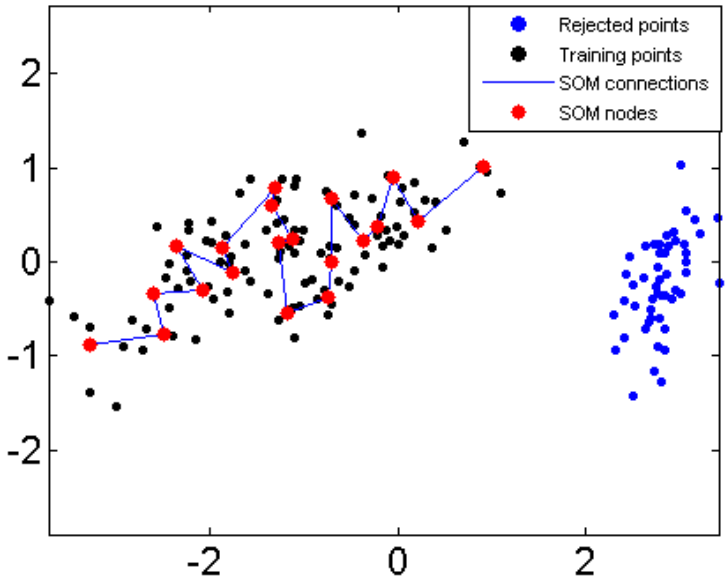

Figure 4.8: SOM_Reject1 with Iris. 1st class left out during training

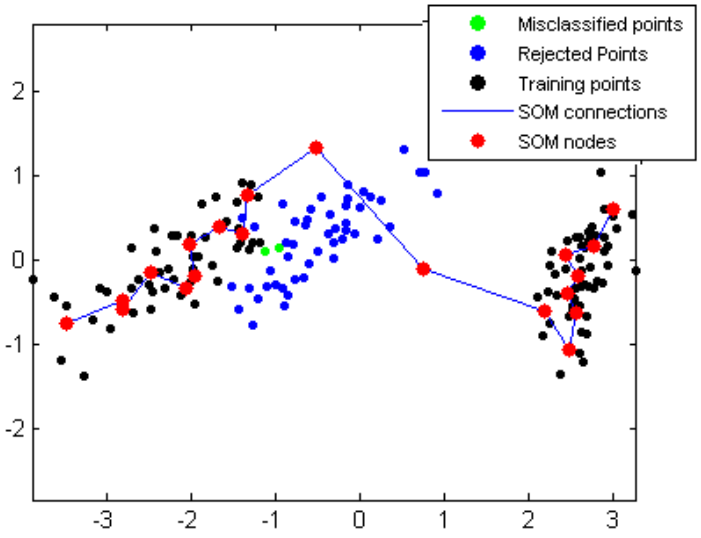

Figure 4.9: SOM_Reject1 with Iris. 2nd class left out during training

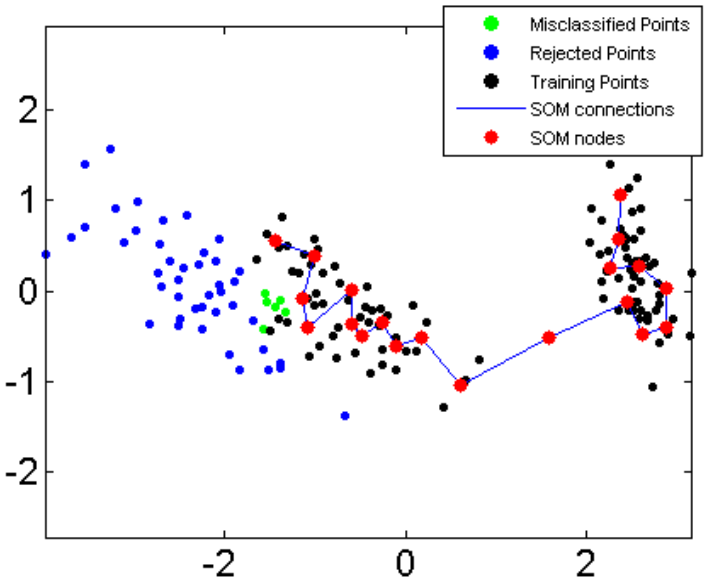

Figure 4.10: SOM_Reject1 with Iris. 3rd class left out during training

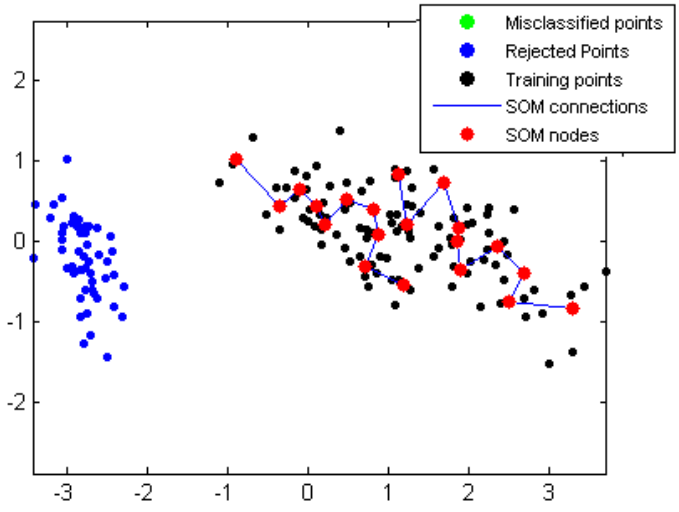

Figure 4.11: SOM_Reject1 single class SOM with Iris. 1st class left out.

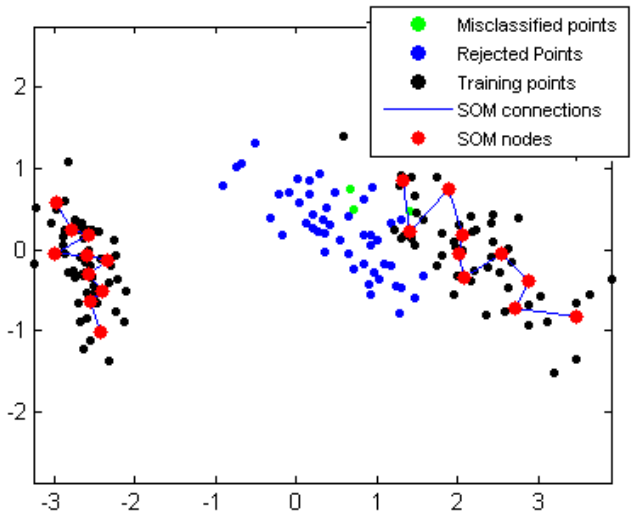

Figure 4.12: SOM_Reject1 single class SOM with Iris. 2nd class left out.

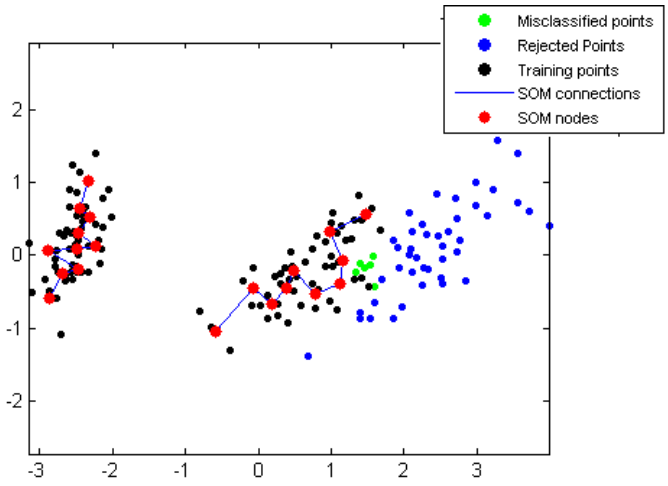

Figure 4.13: SOM_Reject1 single class SOM with Iris. 3rd class left out.

Chapter 5

Classifier with Reject Option using Extreme Value Theory

In this chapter we look at incorporating a reject option in our learning model using Extreme Value Theory (EVT).

5.1 Gaussian Mixture Model

Gaussian Mixture Models (GMMs) are widely used for clustering and classification. It consists of a set of K components, each of which is modeled by a Gaussian distribution parameterized by its mean vector and covariance matrix [20]. Each of the components have a prior probability associated with it. We write the prior probability of the kth component as $\pi(k)$ $\forall k \in \{1, 2, ..., K\}$. The probability density function(pdf) of the kth component is written as $p(\boldsymbol{x}|k)$ and it is just the probability density function of a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{C}_k)$ parameterized by mean $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{C}_k$. The prior probabilities of all the components must sum to 1. Thus, $\sum_{k=1}^{K} \pi(k) = 1$.

For such a mixture, the joint-density is:

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi(k) p(\boldsymbol{x}|k) \tag{5.1}$$

5.1.1 Expectation Maximization

The parameters of a GMM are usually obtained using an algorithm called Expectation Maximization (EM). The parameters of a K component GMM are $(\pi_k, \boldsymbol{\mu}_k, \boldsymbol{C}_k)$ $\forall k \in \{1, 2, ..., K\}$. Let our dataset be $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\} \subset \mathbb{R}^d$. Using Bayes theorem, the probability that a point $\boldsymbol{x}_i$ belongs to component k is given by:

$$p(k|\boldsymbol{x}_i) = \frac{p(\boldsymbol{x}_i|k)\pi(k)}{p(\boldsymbol{x}_i)} \qquad (E) \tag{5.2}$$

Here, $p(\boldsymbol{x}_i)$ is given by equation (5.1). If we now write the log likelihood of the GMM for the training data and differentiate it with respect to the parameters, we get a set of coupled equations for the parameters [18]. The equations giving the parameters for the kth component are given by:

$$\pi(k) = \frac{1}{n} \sum_{i=1}^{n} p(k|\boldsymbol{x}_i) \tag{5.3}$$

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{n} p(k|\boldsymbol{x}_i)\boldsymbol{x}_i}{\sum_{i=1}^{n} p(k|\boldsymbol{x}_i)} \qquad (M) \tag{5.4}$$

$$\boldsymbol{C}_k = \frac{\sum_{i=1}^{n} p(k|\boldsymbol{x}_i)(\boldsymbol{x}_i - \mu_k)(\boldsymbol{x}_i - \mu_k)^T}{\sum_{i=1}^{n} p(k|\boldsymbol{x}_i)} \tag{5.5}$$

We randomly select K distinct data points to use as the initial means $(\boldsymbol{\mu}_k)$. The initial covariance matrices for all the components are diagonal where the $(j, j)th$ element of each matrix is the variance of the jth feature of the data. Next, we compute $p(\boldsymbol{x}_i|k)$ and $\pi(k)$. Now in the Expectation step, we calculate the probability of each point belonging to each cluster $(p(k|\boldsymbol{x}_i)\ \forall i \in \{1, 2, ..., n\}$ and $\forall k \in \{1, 2, ..., K\})$ using equation (5.2). In the Maximization step, we calculate the parameters of the GMM using equations (5.3), (5.4) and (5.5). In this way, we alternately keep performing the Expectation step and the Maximization step until the value of the log likelihood of the data converges.

5.2 Modelling Extreme Values of the Gaussian Mixture Model

A one-sided standard Normal distribution (denoted by $|\mathcal{N}(0,1)|$) has probability density function given by:

$$p(x) = \sqrt{\frac{2}{\pi}} \exp\left(\frac{-x^2}{2}\right) \tag{5.6}$$

Then, the following theorem holds for the extreme values of a one-sided Normal distribution. [8] :

Theorem 2 *Let $\{s_1, s_2, ..., s_n\}$ be independent random variables with a one-sided standard Normal distribution $|\mathcal{N}(0,1)|$. Let $M_n = max\{s_1, s_2, ..., s_n\}$. Then, as $n \to \infty$ the distribution of M_n converges to the Gumbel distribution.*

Using the above result, we can find the distribution of extreme values of multivariate Gaussian random variables. A d-dimensional Gaussian random variable with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{C}$ has pdf given by:

$$p(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{C}|}} \exp\left(\frac{-h(\boldsymbol{x})^2}{2}\right) \tag{5.7}$$

where $h(\boldsymbol{x})$ is the Mahalanobis distance given by

$$h(\boldsymbol{x}) = \sqrt{(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{C}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})} \tag{5.8}$$

The Mahalonobis distance, evaluated with respect to a Gaussian component [20], has a density function given by :

$$p(h(\boldsymbol{x})) = \frac{2}{\sqrt{2\pi}} \exp\left(\frac{-h(\boldsymbol{x})^2}{2}\right) \tag{5.9}$$

This is nothing but the pdf of the one-sided standard normal distribution given by equation (5.6). Thus, using Theorem 2, we can say that the maximum values of the Mahalanobis distance of a Gaussian random variable converges to a Gumbel

distribution. Using this result, we can convert the problem of finding the Extreme Value Distribution (EVD) of a multivariate Gaussian random variable to a univariate case. Instead of analyzing the extreme values of the Gaussian random variable, we just analyze the extreme values of its Mahalanobis distance which is one dimensional.

In a multivariate GMM, there are multiple Gaussian distributions, each with its own Extreme Value Distributions. However, each of their Mahalonobis distances follow Gumbel distribution. Now, for a new input $\boldsymbol{x}$, we can use the technique given in [20] to find its EVD probability. We first calculate its Mahalanobis distance from all the components. Then, we assume that the closest component distribution dominates its EVD probability. This assumption has been previously made by Roberts [20]. The contribution of the other components is assumed to be negligible. In this way, we can find EVD probabilities for a GMM.

5.3 Designing the EVT Based Two Stage Classifier

We now describe our classifier with a reject option. First, for each class in the training set, we train a GMM using the EM algorithm. Thus, we have one GMM per class. Then, for every component in each GMM, we first find the Mahalonobis distances from all the input data belonging to that class (for which the GMM is trained) to the mean of that component. Then, we take the largest 20% of these distances. Using these distances, we estimate the Gumbel location and scale parameters (γ and β) using the maximum likelihood estimation. Similar protocol has been used by other investigators also [20].

We had seen in the previous section that the extreme values of the Mahalanobis distances follow a Gumbel distribution. Thus, for every component of the GMM, we have a corresponding Gumbel distribution of it's extreme values (of Mahalanobis distances). As we had stated in the previous section, we assume that the EVD char-

acteristics of a point is dominated by the component of the GMM with minimum Mahalanobis distance to it. Thus, to find if a point is an extreme value with respect to the GMM, we calculate if it is an extreme value with respect to its closest component of the GMM. The probability of being an extreme value with respect to the closest component is given by the cumulative distribution function (CDF) of its corresponding Gumbel distribution. If this probability is above some pre-determined threshold, it means that the point is an extreme value with respect to the component and hence, it is an extreme value with respect to the GMM. Thus, we reject that point.

Sometimes, the GMM model may not be able to reject all inputs successfully when there are overlapping components [6]. Thus, just like in [22], we train a one-vs-all binary Support Vector Machine (SVM) for every class and fit Weibull distributions to the match and non-match data. We had explained this procedure in detail in Section 3.2. Thus, we have one GMM and one binary one-vs-all SVM for every class. We also have the corresponding EVDs, one from the GMM and one from the SVM classifier for each class. The algorithm is explained in Algorithm 7. We assume there are c classes in our training set with labels $\{l_1, l_2, ...l_c\}$.

Now, for a new input point, for every GMM, we find the component with the minimum Mahalanobis distance from the point. Then, using the Gumbel parameters for that component, we calculate the Gumbel cumulative distribution function for the Mahalanobis distance of the component from the input point. The CDF of a Gumbel distribution with location γ and scale β is given by:

$$P(x) = e^{-e^{-(x-\gamma)/\beta}} \tag{5.10}$$

The Gumbel CDF gives us the probability of the point being an extreme value with respect to that component. The higher this value, the higher the probability of being an extreme value with respect to that component. Thus, if this probability is greater than some sufficiently high pre-determined threshold, we straight away reject the input. For our algorithm, we set this threshold at 0.99. Then, we go to

the corresponding binary Weibull SVM model and calculate the binary SVM score. Based on this, we decide whether to classify or reject this input as explained in Section 3.2 and in Algorithm 2. The whole algorithm is described in Algorithm 8. We call this algorithm **EVT_Reject**.

Algorithm 7: EVT Model Fitting

Data: $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ...(\boldsymbol{x}_n, y_n)\}$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{l_1, l_2, ...l_c\}$

Input:

Pre-trained GMM using EM for each class ;

Pre-trained 1-vs-All binary SVM for each class, with SVM score functions $f_i()$ and support vectors α_i^+, α_i^- for match and non-match data respectively for the ith class $\forall i \in \{1, 2, ..., c\}$;

Tail size multiplier $\theta = 1.5$;

K=No of components of each GMM ;

for *i=1 to c* **do**

  Let the set of binary svm scores be $S_i = \{f_i(\boldsymbol{x}_j)\}$ $(\forall j \in \{1, 2, ..., n\}$ with $y_j = l_i)$;

  Let $q_i^+ = \theta \times |\alpha_i^+|$, $q_i^- = \theta \times |\alpha_i^-|$;

  Let d_i^+ be the smallest q_i^+ positive(match) scores from S_i and d_i^- be the largest q_i^- negative(non-match) scores from S_i ;

  for *k=1 to K* **do**

    $d_{i,k}$=top 20% largest Mahalanobis distances from the mean of kth component($\boldsymbol{\mu}_{i,k}$) to data from ith class ;

    $[\gamma_{i,k},\beta_{i,k}]$=Estimates of Gumbel parameters from $d_{i,k}$;

  end

  $[\nu_{\eta,i},\lambda_{\eta,i},\kappa_{\eta,i}]$=Estimates of Weibull parameters from d_i^+ ;

  $[\nu_{\psi,i},\lambda_{\psi,i},\kappa_{\psi,i}]$=Estimates of Weibull parameters from $-d_i^-$(negate scores to make them positive) ;

end

Output: W_i=$[\gamma_{i,1},\beta_{i,1},\gamma_{i,2},\beta_{i,3},.....,\gamma_{i,K},\beta_{i,K},\nu_{\eta,i},\lambda_{\eta,i},\kappa_{\eta,i},\nu_{\psi,i},\lambda_{\psi,i},\kappa_{\psi,i}]$
$\forall i \in \{1, 2, ..., c\}$

Algorithm 8: EVT_Reject

Input:

Input data point $\boldsymbol{x}$;

Pre-trained GMM for each class $i \in \{1, 2, ..., c\}$;

Pre-trained 1-vs-All binary SVM for each class $i \in \{1, 2, ..., c\}$;

Parameter estimates W_i for each class $i \in \{1, 2, ..., c\}$;

Set $\delta_T = 0.99$ and $\delta_R = 0.5 \times Openness$;

for *i=1 to c* **do**

  Let k=GMM component with minimum Mahalanobis distance from mean $\boldsymbol{\mu}_{i,k}$ to $\boldsymbol{x}$;

  Calculate GMM CDF $P^g_{i,k}(\boldsymbol{x})$ using Gumbel parameters $[\gamma_{i,k},\beta_{i,k}]$ and equation 5.10 ;

  Calculate $P_{\eta,i}(\boldsymbol{x})$ using equation 3.3. ;

  Calculate $P_{\psi,i}(\boldsymbol{x})$ using equation 3.4. ;

  Set t_i=0 ;

  if $P^g_{i,k}(\boldsymbol{x}) < \delta_T$ **then**

    Set t_i=1

  end

  Calculate $T_i(\boldsymbol{x})$ using equation 3.5 ;

end

Find label y using 3.6 or $y = 0$ if $T_i(\boldsymbol{x}) \times t_i < \delta_R \ \forall i \in \{1, 2, ..., c\}$

Output: Label y of $\boldsymbol{x}$ if accepted else reject $\boldsymbol{x}$

Chapter 6

Experimental Results

We now look at the results of SOM_Reject (Algorithm 5) and EVT_Reject (Algorithm 8) Algorithms with the MNIST dataset and compare them the WSVM algorithm of [22] which we had described in Section 3.2.

6.1 MNIST Dataset

The MNIST dataset [16] is a large dataset consisting of images of handwritten digits which is often used for testing classifier algorithms. It was created from the original NIST database of images. The images are of all the digits 0-9. Thus, there are a total of 10 classes of images in the dataset. The images have been size-normalized and centered. Each of the images in the dataset is of size 28×28. Thus, each data point has 784 features. The training set consists of 60,000 images and test set consists of 10,000 images.

We test our algorithms using the protocol described in [22]. We take all the 70,000 images together. We then choose randomly any six labels which become our known classes. Thus, there are four unknown classes. From each of the known classes, we choose 80% of the points which become our training set. We put the rest of the points from the known classes into the test set. In [22], an openness index

is defined to estimate how "open" the problem is. We had stated this in equation (3.7). Here, we vary openness by increasing the number of the unknown classes in the test set from 0 to 4. For openness level 1, we randomly choose one of the four unknown classes. In general for openness level k, $0 \leq k \leq 4$, we randomly choose a set of k unknown classes and check the performance of our system. For each level of Openness, we calculate the accuracy, f-measure, precision and recall. We repeat the entire procedure (that is, selection of six known classes and varying openness level from 0 to 4) ten times. Finally, for each level of openness, we take the average of accuracy, f-measure, precision and recall over the ten folds and report the same.

We train the SVMs for EVT_Reject and WSVM using the LIBSVM library [4]. The SVM parameters are set at the same values as in [22] where C=2 and γ=0.03125. The number of components of the GMM for EVT_Reject could have been chosen using some criteria like Akaike Infromation Criteria(AIC). However, since this is not the main focus of our work, we choose the number of components experimentally. For this, we compare the fmeasure values of EVT_Reject for different levels of openness over five folds with 2, 4, 6, 8 and 10 GMM components (results shown in Table 6.1). Though there is not too much difference between the results, we find that the fmeasure values for the GMM with 4 components are the highest for high levels of openness. Thus, we choose a GMM with 4 components for EVT_Reject.

6.2 Choosing the SOM Rejection Criteria

To choose the rejection criteria for the SOM, we look at the distribution of distances of the points mapped to an individual node of the SOM. We train a SOM with 300 nodes with the MNIST dataset and find the node with the most number of points mapped to it. Then, we find the Euclidean distances of these points from the SOM node. We then normalize the distances (between 0 and 1) and plot the histogram. A typical example is shown in Figure 6.1. We notice that normalized

Table 6.1: MNIST dataset **fmeasure vs openness** for EVT_Reject with different GMM components (denoted by K)

Unknown classes	Openness	$K = 2$	$K = 4$	$K = 6$	$K = 8$	$K = 10$
0	0	98.28	98.41	98.35	98.33	98.38
1	0.0392	91.26	91.71	92.36	91.19	91.40
2	0.0742	87.68	88.08	87.66	87.78	87.71
3	0.1056	86.86	87.15	86.74	86.85	86.88
4	0.1340	85.66	86.08	85.78	85.57	85.66

distances of most of the points from the SOM prototype are centered around 0.5 and are a little more biased towards the maximum distance (1). This is surprising because one would expect most points to be near the SOM prototype (normalized distance $\approx$ 0). Such profiles are observed for a number of nodes. We also find that there exists sufficient space on the $X - axis$ where there are very few or no points. This suggests that thresholding by maximum distance could be too sensitive as many points from different classes could fall inside the decision boundary of the node. Thus, we use Algorithm 6 (SOMReject2) where the rejection threshold is set at $\mu + \sigma$ (where μ and σ are the mean and standard deviation of the distances of points mapped to the node) for individual nodes. We compare the fmeasures of the two rejection criteria for different levels of openness in Table 6.2. We find that SOM_Reject2 indeed performs much better than SOM_Reject1 for higher levels of openness. However, when openness is 0, SOM_Reject2 performs poorly compared to SOM_Reject1 as some legitimate points are rejected by it as expected (higher false negative rate).

We had also trained SOMs with 50 nodes individually for each of the six classes in the training set instead of training a single SOM with 300 nodes on the whole training set. We had then applied our rejection rule on the test data. However, surprisingly we found that the single SOM outperforms the SOMs trained individually. We

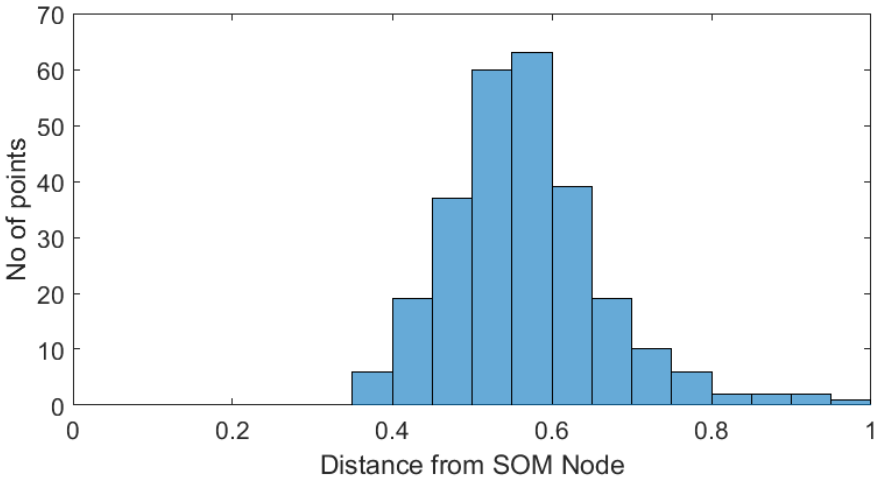

Figure 6.1: Distribution of distances from SOM node. (Total points=266)

found that the individual SOMs have consistently higher false negative rates than the single SOM trained for training set which results in a lower recall and thus, a lower fmeasure. This means that the individual SOMs somehow end up with more conservative boundaries than a single SOM.

Table 6.2: MNIST dataset **fmeasure vs openness** for SOM_Reject1 and SOM_Reject2

Unknown classes	Openness	**SOM_Reject1** (300 nodes, threshold=maximum distance)	**SOM_Reject2** (300 nodes, threshold=$\mu + \sigma$)
0	0	98.78	89.91
1	0.0392	64.12	83.10
2	0.0742	56.29	78.11
3	0.1056	48.23	71.48
4	0.1340	43.06	65.48

6.3 Results and Remarks

Table 6.3: MNIST dataset **accuracy vs openness** for three methods

Unknown classes	Openness	**WSVM** [22]	**SOM_Reject2** (300 nodes, threshold=$\mu + \sigma$)	**EVT_Reject**
0	0	96.83	81.67	96.88
1	0.0392	91.91	80.98	91.76
2	0.0742	91.99	82.18	91.80
3	0.1056	92.38	80.92	92.70
4	0.1340	92.66	79.60	93.03

Table 6.4: MNIST dataset **fmeasure vs openness** for three methods

Unknown classes	Openness	**WSVM** [22]	**SOM_Reject2** (300 nodes, threshold=$\mu + \sigma$)	**EVT_Reject**
0	0	98.39	89.91	98.41
1	0.0392	92.43	83.10	92.29
2	0.0742	89.43	78.11	89.18
3	0.1056	87.10	71.48	87.53
4	0.1340	84.90	65.48	85.51

The variations of accuracy, fmeasure, precision and recall with openness are shown in Tables 6.3, 6.4, 6.5 and 6.6 respectively. We notice some overall trends. We see the EVT_Reject algorithm outperforms WSVM by a small margin for high levels of openness. The SOM_Reject algorithm performs poorly when compared to the other two algorithms with increasing openness. In Table 6.3, we notice a peculiar behaviour. The accuracy of the three algorithms sometimes increases with increasing openness. This is because as openness increases, so does the number of unknown

Table 6.5: MNIST dataset **precision vs openness** for three methods

Unknown classes	Openness	**WSVM** [22]	**SOM_Reject2** (300 nodes, threshold=$\mu + \sigma$)	**EVT_Reject**
0	0	100	100	100
1	0.0392	94.43	85.94	94.48
2	0.0742	89.29	75.87	89.08
3	0.1056	85.28	64.23	86.32
4	0.1340	81.58	55.39	82.97

Table 6.6: MNIST dataset **recall vs openness** for three methods

Unknown classes	Openness	**WSVM** [22]	**SOM_Reject2** (300 nodes, threshold=$\mu + \sigma$)	**EVT_Reject**
0	0	96.83	81.67	96.88
1	0.0392	90.61	81.67	90.31
2	0.0742	89.78	81.67	89.52
3	0.1056	89.22	81.67	88.97
4	0.1340	88.81	81.67	88.51

classes in test set and so does the number of points which our algorithm should reject. If we have a sufficiently low rejection threshold, then our algorithm will be more prone to rejecting points. This pushes up the accuracy. Hence, in openset problems, accuracy is not a very reliable metric. Fmeasure is more indicative of the true performance than accuracy.

We also notice in Tables 6.5 and 6.6 that for SOM_Reject, while recall remains almost constant, the precision decreases pretty steeply. This indicates that the number of false positives increases with openness. Thus, the classifier SOM_Reject ends up including a lot of open space from where points from unknown classes may originate.

Chapter 7

Conclusion and Future Work

In this thesis, we proposed two methods to design classifiers with a reject option. The first method with a SOM is unsupervised and can be used initially to reject points before using another classifier to classify the accepted points. While for low dimensional datasets, it gives good performance, its performance is not so good with the high dimensional MNIST dataset. As we saw, one reason for this could be the SOM nodes end up enclosing large open spaces. As a result, there are too many false positives which bring down the performance. The SOM algorithm might also perform poorly when there is a lot of overlap between classes. Performance with SOM on the MNIST dataset improved when we set a threshold based on mean and variance of the distances rather than maximum distance. This suggests that overall performance of the algorithm will improve if one sets thresholds on the nodes individually by checking the distribution of distances of the points from the SOM nodes.

The second method we proposed is based on using Extreme Value Theory(EVT) with Gaussian Mixture Model and Support Vector Machine. This algorithm gives the best performance with the MNIST dataset. EVT in general shows promising results with the open set problem. It can also be used with other classifiers to improve their performance.

Bibliography

[1] Peter Bartlett and Martin H. Wegkamp. Classification with a reject option using a hinge loss. *Journal of Machine Learning Research*, 9(1):1823–1840, 2008.

[2] Abhijit Bendale and Terrance E. Boult. Towards open set deep networks. In *IEEE Computer Vision and Pattern Recognition(CVPR)*, 2016.

[3] Debrup Chakraborty and Nikhil R. Pal. A novel training scheme for multilayered perceptrons to realize proper generalization and incremental learning. *IEEE Transactions on Neural Networks*, 14(1):1–14, 2003.

[4] C.C. Chang and C.J. Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, 2011.

[5] C.K. Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46, 1970.

[6] David Andrew Clifton, Samuel Hugueny, and Lionel Tarassenko. Novelty detection with multivariate extreme value statistics. *Journal of Signal Processing Systems*, 65:371–389, 2004.

[7] B. Dubuisson and M. Masson. A statistical decision rule with incomplete knowledge about classes. *Pattern Recognition*, 26(1):155–165, 1993.

[8] Paul Embrechts, Claudia Klppelberg, and Thomas Mikosch. *Modelling Extremal Events for Insurance and Finance*. Springer, 1997.

[9] Ana Ferreira and Laurens De Haan. On the block maxima method in extreme value theory: Pwm estimators. *The Annals of Statistics*, 43(1):276–298, 2015.

[10] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1993.

[11] Manfred Gilli and Evis kllezi. An application of extreme value theory for measuring financial risk. *Computational Economics*, 27(2):207–228, 2006.

[12] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Pearson, 2005.

[13] Samuel Kotz and Saralees Nadarajah. *Extreme Value Distributions: Theory and Applications*. World Scientific Publishing Co., 2001.

[14] Khaled Labib and Rao Vemuri. Nsom: A real-time network-based intrusion detection system using self-organizing maps. *Networks and Security*, 2002.

[15] M.R. Leadbetter. On a basis for 'peaks over threshold' modeling. *Statistics and Probability Letters*, 12(4):357–362, 1991.

[16] Yann LeCun, L. Bottou, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324,, 1998.

[17] Russell Muzzolini, Yee-Hong Yang, and Roger Pierson. Classifier design with incomplete knowledge. *Pattern Recognition*, 31(4):345–369, 1998.

[18] Andrew Ng. Cs229 lecture notes. University Lecture Notes.

[19] M. Ramadas, S. Ostermann, and B. Tjaden. Detecting anomalous network traffic with self-organizing maps. In *Recent Advances in Intrusion Detection*, pages 36–54. Springer, 2003.

[20] S.J. Roberts. Novelty detection using extreme value statistics. In *IEE Proceedings - Vision, Image and Signal Processing*, pages 124–129. IEEE, 1999.

[21] JW Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 18(5):401–409, 1969.

[22] Walter J. Scheirer, Lalit P. Jain, and Terrance E. Boult. Probability models for open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2317–2324, 2014.

[23] Walter J. Scheirer, Anderson Rocha, Ross J. Micheals, and Terrance E. Boult. Meta-recognition: The theory and practice of recognition score analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1689–1695, 2011.

[24] Walter J. Scheirer, Anderson Rocha, Archana Sapkota, and Terrance E. Boult. Towards open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(10):1757–1772, 2012.

[25] Bernhard Scholkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[26] David M.J. Tax and Robert P.W. Duin. Support vector data description. *Machine Learning*, 54:45–66, 2004.