HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES

MASTER THESIS

Herr
Nana Abeka Otoo

Determining of Classification Label Security/Certainty

2022

Faculty of **Applied Computer and Life Sciences**

MASTER THESIS

Determining of Classification Label Security/Certainty

Author:
Nana Abeka Otoo

Study Programme:
Applied Mathematics for Network and Data Sciences

Seminar Group:
MA18w1-M

First Referee:
Prof. Dr. Thomas Villmann

Second Referee:
MSc. Jensun Ravichandran

Mittweida, 2022

A big thank you goes to

my loved ones
for their support and love.

Special gratitude goes to

Prof. Dr. Thomas Villmann
for his supervision and guidance
and MSc. Jensun Ravichandran
for his supervision, guidance and comments

Bibliographic Information

Otoo, Nana Abeka: Determining of Classification Label Security/Certainty, 53 pages, 19 figures, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer and Life Sciences

Master Thesis, 2022

Abstract

Classification label security determines the extent to which predicted labels from classification results can be trusted. The uncertainty surrounding classification labels is resolved by the security to which the classification is made. Therefore, classification label security is very significant for decision-making whenever we are encountered with a classification task. This thesis investigates the determination of the classification label security by utilizing fuzzy probabilistic assignments of Fuzzy c-means. The investigation is accompanied by implementation, experimentation, visualization and documentation of the results.

I. Contents

II. List of Figures

III. List of Tables

IV. Preface

1 Introduction

Machine learning as a field of study has gained prominence and publicity in academia and industry in recent times. One is not wrong to say that, Machine learning has become a significant matter of discussion among students, industry players and every professional whose work, one way or the other, is influenced by it. We can at this stage realize why almost every practical process witnessed in our lives today is either applying machine learning or is migrating to its adoption. The benefits that arise with the utilization of machine learning processes can be exemplified and witnessed in areas such as medicine, security, engineering, commerce, agriculture, only to mention a few since the list keeps growing with new innovations and methods being added day by day.

In this regard, a learning machine is a model tasked to learn from a given data and make valuable predictions on new data that was not used in the learning process. A well-grounded area in machine learning is prototype-based models that learn prototypes from a given data set by training and make classifications on new data using the difference between the data points and learned prototypes. It is suitable to observe the many prototype-based algorithms that are commonly applied today in most processes. A family of prototype-based models that have pitched the interest of most users is the well-known Learning Vector Quantization. The interest in Learning Vector Quantization can be explained by the facts that surround its easily comprehensible theoretical considerations and practical implementations. At the moment, Learning Vector Quantisation holds an enviable position among the classification algorithms found in the area of prototype-based models. We begin to ask for reasons in this regard; a simple answer to this question lies in its understandable mathematical inclination, easy usability, high performance coupled with outcomes that can be explained. It is right to say that every classifier has the primary duty of making reasonable or, so to say, good classifications. Again, it is desirable from the usage point of view that a good classifier possesses the attribute that allows users to know the degree to which classification results can be trusted. The ability of a classifier to come equipped with such an attribute is very significant because it provides the security to which classification labels can be accepted. The classification label security remains vital for making decisions in this regard.

1.1 Motivation

T. Kohonen introduced Learning Vector Quantisation (LVQ) as a prototype-based analog of unsupervised competitive learning, which he designed to classify different patterns in data [2]. Even though LVQ results in optimal reference vectors, it is characterized by issues of divergent reference vectors [3]. This challenge, among others, led to attempts geared towards improved variants in [2] but to no avail. The outcomes of these variants are, in practice, not the same [4].

The introduction of Generalized Learning Vector Quantization (GLVQ) by Sato and Yamada solved the problem concerning the diverging reference vectors, utilizes a cost function-based approach, and incorporates convergence conditions in the winner takes all learning rule [3]. The reliability and robustness of LVQ and its variants is penchant on the homogeneity of data used and, most importantly, they heavily utilize and depend on Euclidean distance measure which may not be universal for all cases understudy [5].

GLVQ provides a good generalization with convergence conditions, based on any standard distance metric which can be optimized [6]. A substantial and balanced step for solving this problem led to applying relevant factors to specify a family of distance measures leading to the Relevance GLVQ [7].

A variant of Relevance LVQ called Matrix Relevance LVQ utilizes a matrix of relevances that will be learned in the same manner as the weights using GLVQ update rules [8]. It remains to show which choice of a matrix of relevant factors initialization is required to parametrize the distance measure for optimal classification results [7, 9]. Consider the optimal classification results linked with the certainty/security of the classification labels from a Fuzzy clustering utilizing a covariance matrix [10]. A version of LVQ which utilizes cross-entropy for classification is introduced [11, 12]. The use of cross-entropy optimization in LVQ is discovered to result in class positions that ensure classification label security [11]. The computation of classification label security is reliant on the converged reference vectors [1] whose optimization, in turn is also dependent on its initialization [13]. The classification label security remains to be investigated in this regard. Consider unsupervised Fuzzy c means (FCM) by Bezdek, which utilizes fuzzy membership to ascertain the certainty of cluster members [1]. A good way forward is to investigate the utilization of fuzzy probabilistic assignments of FCM to determine the classification label security with applications to GLVQ, Generalized Matrix Learning Vector Quantization (GMLVQ) and Cross-Entropy Learning Vector Quantization (CELVQ).

1.2 Brief on Clustering

The clustering task involves partitioning data without labels into subgroups based on data features representing structure in the data set. The underlying similarity between data patterns is used for arranging data into clusters. We consider the following definitions:

Definition 1.1 [1] *Hard c-Partition.* $X = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n\}$ is any finite set; V_{cn} is the set of real $c \times n$ matrices; c is an integer, $2 \leq c < n$. *Hard c-partition space for* X is the set

$$M_c = \left\{ U \in V_{cn} \;\middle|\; u_{ik} \in \{0,1\} \,\forall\, i,k\,;\, \sum_{i=1}^{c} u_{ik} = 1 \,\forall\, k\,;\, 0 < \sum_{k=1}^{n} u_{ik} < n \,\forall\, i \right\}$$

$$u_{ik} \in \{0,1\}, \quad 1 \le i \le c, \quad 1 \le k \le n \tag{1.1a}$$

$(1.1a)$ means that the fuzzy probabilistic assignment of the ith partition of X is 1 or 0 when $\mathbf{x}_k$ is in the ith partition and otherwise respectively [1].

$$\sum_{i=1}^{c} u_{ik} = 1, \quad 1 \le k \le n \tag{1.1b}$$

$(1.1b)$ indicates each pattern $\mathbf{x}_k$ can be uniquely assigned a cluster c subsets [1].

$$0 < \sum_{k=1}^{n} u_{ik} < n, \qquad 1 \le i \le c \tag{1.1c}$$

$(1.1c)$ indicates there should be at least two partition subsets of X and these subsets should also be less than the cardinality of X: $2 \le c < n$ [1].

Definition 1.2 [1] *Fuzzy c-Partition.* X is any finite set; V_{cn} is the set of real $c \times n$ matrices; c is an integer, $2 \le c < n$. *Fuzzy c-partion space for* X is the set

$$M_{fc} = \left\{ U \in V_{cn} \;\middle|\; u_{ik} \in [0,1] \,\forall\, i,k \,;\, \sum_{i=1}^{c} u_{ik} = 1 \,\forall\, k\,;\, 0 < \sum_{k=1}^{n} u_{ik} < n \,\forall\, i \right\} \tag{1.2}$$

condition (1.1a) is extendend to include all values between 1 and 0. Hence removing the crisp assignments of membership functions u_{ik} [1].

Definition 1.3 [14] *Possibilistic c-Partition.* X is any finite set; V_{cn} is the set of real $c \times n$ matrices; c is an integer, $2 \le c < n$. *Possibilistic c-partion space for* X is the set

$$M_{pc} = \left\{ U \in V_{cn} \;\middle|\; u_{ik} \in [0,1] \,\forall\, i,k\,;\, \forall\, k\, \exists\, i \ni u_{ik} > 0 \right\} \tag{1.3}$$

the column condition in (1.1b) is changed and replaced with $0 < \sum_{i=1}^{c} u_{ik} \le c$ and u_{ik} is referred to as tipicality of data pattern $\mathbf{x}_k$ [15].

2 Objective Function Clustering

The primary approach here is to utilize a sum of squares errors function optimized to achieve a minimized error point at which clustering results can be accepted. It is significant to know that optimal clustering, in this case, is achieved at the local extrema of the objective function [1].

2.1 Fuzzy c-Means

As described by Bezdek [1], the Fuzzy c-means provides a soft alternative to the Hard c-means clustering algorithm. The discrepancy comes from the way the fuzzy U- matrix is partitioned along with some conditions allowing the crisp assignments as seen in hard c- means to now include the full range of probabilistic assignments as defined above in (1.2) and referred to as fuzzy memberships. The fuzzy memberships determine the degrees to which patterns belong in a partition(cluster).

Theorem 2.1 *[1] let the objective function of Fuzzy c-Means be*

$$J_m(U,\mathbf{v}) = \sum_{k=1}^{n}\sum_{i=1}^{c}(u_{ik})^m (d_{ik})^2$$

and assume an inner product norm to be

$$\begin{aligned}(d_{ik})^2 &= \| \mathbf{x}_k - \mathbf{v}_i \|_A^2 \\ &= \langle \mathbf{x}_k - \mathbf{v}_i, \mathbf{x}_k - \mathbf{v}_i \rangle_A \\ &= (\mathbf{x}_k - \mathbf{v}_i)^T A (\mathbf{x}_k - \mathbf{v}_i)\end{aligned}$$

where

$$U \in M_{fc}$$

is the fuzzy c- partion of X and

$$\mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c) \in \mathbb{R}^{cp} \text{ with } \mathbf{v}_i \in \mathbb{R}^p$$

is the cluster center or prototypes of u_i, $1 \le i \le c$

$$\text{choose } m \in (1,\infty)$$

let X have at least c less than n distinct points, and define for all k the sets

$$I_k = \{ i \mid 1 \le i \le c;\ d_{ik} = \| \mathbf{x}_k - \mathbf{v}_i \|_A = 0 \}$$

$$\tilde{I}_k = \{1, 2, \ldots, c\} - I_k$$

then $J_m(U, \mathbf{v})$ *may be globally minimised only if*

$$I_k = \varnothing \Rightarrow u_{ik} = \frac{1}{\left[\sum_{j=1}^{c}\left(\frac{d_{ik}}{d_{jk}}\right)^{\frac{2}{(m-1)}}\right]} \tag{2.1a}$$

or

$$I_k \neq \varnothing \Rightarrow u_{ik} = 0 \; \forall \; \tilde{I}_k \quad and \quad \sum_{i \in I_k} u_{ik} = 1 \tag{2.1b}$$

$$\mathbf{v}_i = \frac{\sum_{k=1}^{n}(u_{ik})^m \mathbf{x}_k}{\sum_{k=1}^{n}(u_{ik})^m} \; \forall \; 1 \leq i \leq c \tag{2.1c}$$

Figure 2.1: FCM Algorithm, Bezdek [1]

Store	Unlabelled Object Data $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\} \subset \Re^p$
	$*1 < c < n$ $*m > 1$ $*l_{max} = iteration\ limit$ $pick * Norm\ for\ J_m : \|\|\mathbf{x}\|\|_A = \sqrt{\mathbf{x^T A x}}$ $*0 < \varepsilon = termination\ criterion$
	$Initialize\ U^0 \in M_{fc}(1.2)\ at\ iteration\ l, l = 0, 1, 2, \ldots, :$
Do	$Calculate\ the\ cluster\ centers\ v_i^l\ using(2.1c)\ and\ U^l$ $update\ U^l\ with\ (2.1a)(2.1b)\ and\ v_i^l$ $Compare\ U^l\ to\ U^{l+1}\ in\ a\ convenient\ matrix\ norm : if\ \parallel U^{l+1} - U^l \parallel \leq \varepsilon\ stop :$ $otherwise\ return\ to\ step\ 1.$

The parameter $[0 < \varepsilon \ll 1]$ in Figure 2.1 must be chosen to be very small. The fuzzifier(m) must be chosen cautiously to fit the data. The equation$(2.1b)$ is used to account for the scarce occurrance of singularity$(\mathbf{x}_k = \mathbf{v}_i)$ where $d_{ik} = 0$, u_{ik} assignments are spread over $\mathbf{v}_i$ and u_{ik} in d_{ik} greater than 0 automatically become 0's [14]. It must be noted,

$$\lim_{m \to 1^+}\{u_{ik}\} = \begin{cases} 1, & d_{ik} < d_{jk} \; \forall \; j \neq i \\ 0, & \text{otherwise} \end{cases} \tag{2.2a}$$

and consequently,

$$\begin{aligned} \lim_{m \to 1^+}\left\{\left(\mathbf{v}_i = \frac{\sum_{k=1}^{n}(u_{ik})^m \mathbf{x}_k}{\sum_{k=1}^{n}(u_{ik})^m}\right)\right\} \\ = \frac{\sum_{k \in i}\mathbf{x}_k}{n_i} \\ = \tilde{\mathbf{v}}_i; \quad 1 \leq i \leq c \end{aligned} \tag{2.2b}$$

$(2.2a)$ and $(2.2b)$ shows that cluster centroid moves closer to the general mean and hence FCM become crisply assigned with $u_{ik} = \{0,1\}$ which results in Hard c-means (HCM) [1]. This reason accounts for the choice of m.

3 Learning Vector Quantization

3.1 Introduction to Learning Vector Quantization

T. Kohonen introduced Learning Vector Quantization(LVQ) as a prototype-based supervised learning model with the characteristics of being robust and intuitive [2]. LVQ presents an improvement to the nearest neighbor classifiers by introducing prototypes vectors that are learned and optimized to give improved results in classification [12]. Even though LVQ is characterized by producing optimal borders, it suffers a weakness of being heuristically inclined, and also, the instability reference vectors become a matter of concern in its application to most classification tasks [2,5].

Given a training set $X = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_N\} \subseteq \mathbb{R}^n$ with its class labels $c(\mathbf{x}) \in \mathscr{C} = \{1, 2, \ldots, C\}$, we define a prototype set of vectors $W = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_M\} \subseteq \mathbb{R}^n$ such that every $\mathbf{w} \in W$ has a corresponding class $c(\mathbf{w}) \in \mathscr{C}$. The training of prototype vectors is based on a competitive learning known as winner-takes-all rule(3.1) until the prototypes vectors become typical of the classes they represent.

$$S(\mathbf{x}) = \arg\min_k \, d(\mathbf{x}, \mathbf{w}_k), 1 \leq k \leq M \tag{3.1}$$

Consider data point $\mathbf{x}$, the protype vectors $\mathbf{w}_{s(\mathbf{x})}$ is strengthened if $c(\mathbf{x}) = c\left(\mathbf{w}_{s(\mathbf{x})}\right)$ and weakened if $c(\mathbf{x}) \neq c\left(\mathbf{w}_{s(\mathbf{x})}\right)$ based on an update rule defined in (3.3) utilising (3.2) and a small but positive learning rate η

$$\psi\left(c(\mathbf{x}), c\left(\mathbf{w}_{s(\mathbf{x})}\right)\right) = \begin{cases} +1, & c(\mathbf{x}) = c\left(\mathbf{w}_{s(\mathbf{x})}\right) \\ -1, & c(\mathbf{x}) \neq c\left(\mathbf{w}_{s(\mathbf{x})}\right) \end{cases} \tag{3.2}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta\, \psi(\mathbf{x} - \mathbf{w}_t); \quad \mathbf{w}_t = \mathbf{w}_{s(\mathbf{x})}; \quad 0 < \eta \ll 1 \tag{3.3}$$

Though the standard Euclidean distance $d(\mathbf{x}, \mathbf{w}_k)$ is primarily utilized in LVQ, it is not limited to it, and that any standard dissimilarity measure is allowed if it fits the data set in question [16]. The heuristic inclination and the problem of instability of reference vectors led to the development of many LVQ variants [2]. A more mathematically inclined and generalized version is introduced by Sato and Yamada, which solved the aforementioned problems of LVQ [3].

3.2 Generalized Learning Vector Quantization

Sato and Yamada successfully present a generalized version of the LVQ variants, which employs the use of a cost function and an update rule that incorporates convergence conditions for prototype vectors [3].

Let d be any differentaible dissimilarity measure, $\mathbf{w}^{+}$ is the best matching correct prototype vector if $c(\mathbf{x}) = c\left(\mathbf{w}_{s(\mathbf{x})}\right)$ and $\mathbf{w}^{-}$ be the best matching incorrect prototype vector if $c(\mathbf{x}) \neq c\left(\mathbf{w}_{s(\mathbf{x})}\right)$ then a function $\mu(\mathbf{x})$ referred to as the classifier function is

$$\mu(\mathbf{x}) = \frac{d\left(\mathbf{x}, \mathbf{w}^{+}\right) - d\left(\mathbf{x}, \mathbf{w}^{-}\right)}{d\left(\mathbf{x}, \mathbf{w}^{+}\right) + d\left(\mathbf{x}, \mathbf{w}^{-}\right)}$$

$\mu(\mathbf{x}) \in [-1, 1]$, indicating $d\left(\mathbf{x}, \mathbf{w}^{+}\right) < d\left(\mathbf{x}, \mathbf{w}^{-}\right)$ whenever classification is correct meaning $\mu(\mathbf{x})$ is negative and incorrect classification indicates $\mu(\mathbf{x})$ is positive. The cost function is given by,

$$J_{GLVQ}(X, W) = \sum_{i=1}^{n} f\left(\mu\left(\mathbf{x}_i\right)\right) \tag{3.4}$$

The non-linear activation function f, which increases monotonically, is usually chosen as the sigmoid function

$$f_t(\mathbf{x}) = \frac{1}{1 + e^{\frac{-\mathbf{x}}{t}}}; \quad t > 0$$

minimization of the cost function in (3.4) is done using the stochastic gradient descent Learning (SGDL), and the update rules are given by (3.7)

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{w}^{+}} &= \frac{\partial f}{\partial \mu} \cdot \frac{\partial \mu}{\partial d^{+}(\mathbf{x})} \cdot \frac{\partial d^{+}(\mathbf{x})}{\partial \mathbf{w}^{+}} \\ &= \frac{\partial f}{\partial \mu} \cdot \frac{-2 d^{-}(\mathbf{x})}{\left(d^{+}(\mathbf{x}) + d^{-}(\mathbf{x})\right)^2} (-2)\left(\mathbf{x} - \mathbf{w}^{+}\right) \end{aligned} \tag{3.5}$$

Similarly,

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{w}^{-}} &= \frac{\partial f}{\partial \mu} \cdot \frac{\partial \mu}{\partial d^{-}(\mathbf{x})} \cdot \frac{\partial d^{-}(\mathbf{x})}{\partial \mathbf{w}^{-}} \\ &= \frac{\partial f}{\partial \mu} \cdot \frac{2 d^{+}(\mathbf{x})}{\left(d^{+}(\mathbf{x}) + d^{-}(\mathbf{x})\right)^2} (-2)\left(\mathbf{x} - \mathbf{w}^{-}\right) \end{aligned} \tag{3.6}$$

from (3.5) and (3.6) we have the update rule (3.7)

$$\Delta \mathbf{w}^{\pm} \propto \frac{-\partial f}{\partial \mu} \cdot \frac{\pm 2 d^{\mp}(\mathbf{x})}{\left(d^{+}(\mathbf{x}) + d^{-}(\mathbf{x})\right)^2} \cdot \frac{\partial d\left(\mathbf{x}, \mathbf{w}^{\pm}\right)}{\partial \mathbf{w}^{\pm}} \tag{3.7}$$

form Equations (3.5) and (3.6) we identify that the attraction and repulsion scheme used in LVQ is also preserved in GLVQ [16] . However, it must be noted that the dissimilarity measure employed in (3.7) is the squared Euclidean distance [3].

3.3 Generalized Matrix Learning Vector Quantization

GLVQ provides a conceptual framework for which all generalized LVQ could be developed. The GLVQ requirement of using a differentiable dissimilarity measure chosen as the standard Euclidean distance in [3] is not ideal for all problems [16]. The search for a dissimilarity that can work well for different data sets while keeping the generalization requirement of differentiability led to the introduction of Generalized Relevance Learning Vector Quantization (GRLVQ) [5]. The dissimilarity measure used in GRLVQ is specified with relevant factors, which are learned in the same manner as prototypes in GLVQ [5]. An advanced variant of the GRLVQ, which utilizes a full matrix of relevances in specifying the dissimilarity measure used in GLVQ, is introduced and referred to as Generalized Matrix Learning Vector Quantization (GMLVQ) [5].

The dissimilarity measure in matrix-GLVQ is given by

$$d_{\Omega}(\mathbf{x},\mathbf{w}) = (\mathbf{x}-\mathbf{w})^{T}\,\Omega^{T}\Omega\,(\mathbf{x}-\mathbf{w});\quad \Omega \in \mathbb{R}^{m\times n},$$

when m is same as n ; matrix $\Lambda = \Omega^{T}\Omega \in \mathbb{R}^{n\times n}$; Ω serves the purpose of a projection matrix [16]

$$d_{\Omega}(\mathbf{x},\mathbf{w}) = (\Omega(\mathbf{x}-\mathbf{w}))^{2} \tag{3.8}$$

with a positive definite matrix Λ, (3.8) can be taken as the Euclidean distance. Given a classifier of the form

$$\mu(\mathbf{x}) = \frac{d_{\Omega}(\mathbf{x},\mathbf{w}^{+}) - d_{\Omega}(\mathbf{x},\mathbf{w}^{-})}{d_{\Omega}(\mathbf{x},\mathbf{w}^{+}) + d_{\Omega}(\mathbf{x},\mathbf{w}^{-})}$$

The extent of classification security is based on the level to which $d_{\Omega}(\mathbf{x},\mathbf{w}^{+}) < d_{\Omega}(\mathbf{x},\mathbf{w}^{-})$ [5]. The cost function is given by

$$J_{GMLVQ}(X,W) = \sum_{i=1}^{n} f(\mu(\mathbf{x}_i)) \tag{3.9}$$

Just like in GLVQ, the weights updation in (3.11) and the matrix adaptation in (3.10) is done simultaneously [8] with the SGDL used in minimization of (3.9)

$$\Delta\Omega \propto \frac{-\partial f}{\partial \mu}\left(\frac{\partial \mu}{\partial d_{\Omega}^{+}(\mathbf{x})}\cdot\frac{\partial d_{\Omega}^{+}(\mathbf{x})}{\partial \Omega} + \frac{\partial \mu}{\partial d_{\Omega}^{-}(\mathbf{x})}\cdot\frac{\partial d_{\Omega}^{-}(\mathbf{x})}{\partial \Omega}\right) \tag{3.10}$$

$$\Delta\mathbf{w}^{\pm} \propto \frac{-\partial f}{\partial \mu}\cdot\frac{\pm 2 d_{\Omega}^{\mp}(\mathbf{x})}{\left(d_{\Omega}^{+}(\mathbf{x}) + d_{\Omega}^{-}(\mathbf{x})\right)^{2}}\cdot\frac{\partial d_{\Omega}(\mathbf{x},\mathbf{w}^{\pm})}{\partial \mathbf{w}^{\pm}} \tag{3.11}$$

3.4 Cross-Entropy in Learning Vector Quantization

We refer to the same introduction and parameters as used in GLVQ. Considering an information theoretic approach, the training set employed in the learning process comes along with probabilistic target class information given by $(X,T) = \{\mathbf{x}_i, \mathbf{t}_i\}_{i=1}^{N}$ with $\mathbf{t}_i$ being the probabilistic class targets satisfying the conditions $t_{ij} \in [0,1]$ and $\sum_j t_{ij} = 1$ [11].

Given a data point $\mathbf{x} \in X$, consider the class probability vector $p(\mathbf{x}) = (p_1(\mathbf{x}), \ldots, p_C(\mathbf{x}))^T$. Assume a model class predictor $p_W(\mathbf{x}) = (p_W(1|\mathbf{x}), p_W(2|\mathbf{x}), \ldots, p_W(C|\mathbf{x}))^T$ by Soft Learning Vector Quantization(SLVQ) analogy using model parameters from set W. The objective here is to clearly maximize the mutual information between $p(\mathbf{x})$ and $p_W(\mathbf{x})$ by minimizing the divergence between them [11]. Hence, a function that represents this divergence is,

$$L(X,W) = D_{KL}(p(\mathbf{x})||p_W(\mathbf{x})) \tag{3.12}$$

where the Kulbach-Liebler divergence is,

$$D_{KL}(p(\mathbf{x})||p_W(\mathbf{x})) = H(p(\mathbf{x})) - Cr(p(\mathbf{x}), p_W(\mathbf{x}))$$

$H(p(\mathbf{x}))$ indicates Shanon entropy and $Cr(p(\mathbf{x}), p_W(\mathbf{x}))$ indicates cross-entropy. It must be noted that alternative divergence such as the Renyi-α-divergence may also be considered in this regard,

$$D_\alpha(p(\mathbf{x})||p_W(\mathbf{x})) = \frac{1}{1-\alpha}\log\left(\sum_k (p_k(\mathbf{x}))^\alpha \cdot (p_W(k|\mathbf{x}))^{1-\alpha}\right) \tag{3.13}$$

as $\alpha \to 1$ we have that

$$D_\alpha(p(\mathbf{x})||p_W(\mathbf{x})) \to D_{KL}(p(\mathbf{x})||p_W(\mathbf{x}))$$

because the Shanon entropy is independent of the learning parameters, the local errors in (3.12) is minimized, taking into consideration only the cross-entropy

$$\frac{\partial}{\partial w} D_{KL}(p(\mathbf{x})||p_W(\mathbf{x})) = \frac{-\partial}{\partial w} Cr(p(\mathbf{x}), p_W(\mathbf{x})) \tag{3.14}$$

3.4.1 Soft Learning Vector Quantization

The primary aim is to model a soft class predictor that follows conventional learning vector quantization (prototype-based with Euclidean dissimilarity measure) [11,12,17].

Hence given $\mathbf{x} \in X$, the probability density is determined by

$$P_W(\mathbf{x}) = \sum_{j=1}^{N} p\left(\mathbf{x}|\mathbf{w}_j\right) p\left(\mathbf{w}_j\right) \tag{3.15}$$

where for prototype $\mathbf{w}_j \in W$, $p\left(\mathbf{w}_j\right)$ indicates the prior probability and $p\left(\mathbf{x}|\mathbf{w}_j\right)$ indicates the probability of prototype $\mathbf{w}_j$ to induce $\mathbf{x}$. We incorporate fixed classes $c \in \mathscr{C}$ of the data point together with LVQ principles of best correct matching prototypes and best incorrect matching prototypes arriving at a joint probability density of the form

$$P_W(\mathbf{x}, c) = \sum_{j:c\left(\mathbf{w}_j\right)=c} p\left(\mathbf{x}|\mathbf{w}_j\right) p\left(\mathbf{w}_j\right) \tag{3.16}$$

and

$$P_W(\mathbf{x}, \neg c) = \sum_{j:c\left(\mathbf{w}_j\right)\neq c} p\left(\mathbf{x}|\mathbf{w}_j\right) p\left(\mathbf{w}_j\right) \tag{3.17}$$

referred to as the probability that $\mathbf{x}$ is induced by a mixture of Gaussians with the correct class and the probability that $\mathbf{x}$ is induced by a mixture of Gaussians with the incorrect class, respectively [11, 17]. Concerning Soft Learning Vector Quantization (SLVQ), the cost function minimized by stochastic gradient descent learning is given by

$$L_{SLVQ}(X, W) = -\sum_{k} \ln\left(\frac{P_W(\mathbf{x}_k, c_k)}{P_W(\mathbf{x}_k, \neg c_k)}\right) \tag{3.18}$$

and for Robust Soft Learning Vector Quantisation (RSLVQ),

$$L_{RSLVQ}(X, W) = -\sum_{k} \ln\left(\frac{P_W(\mathbf{x}_k, c_k)}{P_W(\mathbf{x}_k)}\right) \tag{3.19}$$

where,

$$P_W(\mathbf{x}_k) = P_W(\mathbf{x}_k, c_k) + P_W(\mathbf{x}_k, \neg c_k) \tag{3.20}$$

In line with Seo and Obermayer [17], $P_W(\mathbf{x}_k)$ solves the problem of instability encountered whenever infinity is attained by the cost function in (3.18). The updates of prototypes for SLVQ is done using

$$\Delta\mathbf{w} \propto \frac{-\partial}{\partial\mathbf{w}_l} L_{SLVQ}(X, W)$$

and in the case of RSLVQ,

$$\Delta\mathbf{w} \propto \frac{-\partial}{\partial\mathbf{w}_l} L_{RSLVQ}(X, W)$$

3.4.2 Robust Soft Learning Vector Quantization with Cross-Entropy Optimization

GLVQ, including its variants together with many other prototype-based classifiers, are generally accepted to be highly robust and involve the optimization of the classification error to attain classification results that are highly intrepretable [12]. A version of LVQ which utilizes the cross-entropy maximization, motivated by information-theoretic principles, is introduced as a generalization of RSLVQ [11]. Hence the cost function of the form,

$$E(X,W) = \sum_{\mathbf{x}} D_{KL}(t(\mathbf{x})||p_W(\mathbf{x}))$$

from the cross-entropy in (3.14). Considering a relation of this model based on prototypes $W = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N\}$ and class responsibilities $c(\mathbf{w}_k)$ we have,

$$Cr_W(\mathbf{x}) = \sum_{c=1}^{C} t_c(\mathbf{x}) \cdot \log(p_W(c|\mathbf{x}))$$

and using the model class prediction probability from SLVQ,

$$p_W(c|\mathbf{x}) = \frac{P_W(\mathbf{x},c)}{P_W(\mathbf{x})}$$

with

$$\begin{aligned} P_W(\mathbf{x},c) &= \sum_{j:c(\mathbf{w}_j)=c} \exp\left(-d_\Omega\left(\mathbf{x},\mathbf{w}_j\right)\right) \\ &= \sum_{j:c(\mathbf{w}_j)=c} \exp\left(-\left(\Omega\left(\mathbf{x}-\mathbf{w}_j\right)\right)^2\right) \end{aligned}$$

and

$$\begin{aligned} P_W(\mathbf{x}) &= \sum_{l} \exp(-d_\Omega(\mathbf{x},\mathbf{w}_l)) \\ &= \sum_{l} \exp\left(-(\Omega(\mathbf{x}-\mathbf{w}_l))^2\right) \end{aligned}$$

we indicate that, $d_\Omega(\mathbf{x},\mathbf{w}_j)$ for all $\mathbf{w}_j \in W$ follows the analogy of the dissimilarity measure utilized in GMLVQ. We have the cross-entropy presented as

$$Cr_W(\mathbf{x}) = \sum_{c=1}^{C} t_c(\mathbf{x}) \cdot \log\left(\frac{P_W(\mathbf{x},c)}{P_W(\mathbf{x})}\right) \tag{3.21}$$

From the results in (3.21), is a generalization RSLVQ [11]. Concerning mutually exclusive training data, the cost function approaches RSLVQ cost function [11]. We account for this by considering $t_{ij} \in \{0,1\}$ together with $\sum_j t_{ij} = 1$, when we assume the target probability accross the classes is mutually exclusive. So considering one prototype per class, $t_c(\mathbf{x}) = 1$ in (3.21) arriving at the same cost functon (3.19) for RSLVQ. Mathematically, we have

$$p(t_i|\mathbf{x}_i) = \prod_{j=1}^{C} p_j(\mathbf{x}_i)^{t_{ij}}$$

and

$$p(c_i|\mathbf{x}_i) = \prod_{j=1}^{C} (p_W(j,\mathbf{x}_i))^{t_{ij}}$$

referred to as the true conditional target probabilty for $\mathbf{x}_i$ and model target conditional probability for $\mathbf{x}_i$ respectively expressed as multinomial distributions [11]. We further consider the log-likelihood ratio

$$\log \frac{p(T|X)}{p_W(C|X)} = \log\left(\prod_{i=1}^{N} \frac{p(t_i|\mathbf{x}_i)}{p_W(c_i|\mathbf{x}_i)}\right)$$

expanded as

$$\begin{aligned} &= \sum_{i=1}^{N} \log(p(t_i|\mathbf{x}_i)) - \sum_{i=1}^{N} \log(p_W(c_i|\mathbf{x}_i)) \\ &= \sum_{i=1}^{N}\sum_{j=1}^{C} t_{ij} \log\left(p_j(\mathbf{x}_i)\right) - \sum_{i=1}^{N}\sum_{j=1}^{C} t_{ij} \log(p_W(j|\mathbf{x}_i)) \end{aligned}$$

and we have the form observed in (3.12) below

$$= \sum_{i=1}^{N} H(t_i) - Cr(t_i, p_W(\mathbf{x}_i))$$

The cross-entropy is minimized for gradient descent learning with respect to parameter W as $\frac{\partial}{\partial \mathbf{w}_l} Cr_W(\mathbf{x})$ and the prototype updates are done using,

$$\Delta \mathbf{w} \propto \frac{-\partial}{\partial \mathbf{w}_l} Cr_W(\mathbf{x}) \tag{3.22}$$

3.5 Classification Label Security/Certainty

We consider a training set $X = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_N\} \subseteq \mathbb{R}^n$ with its class labels $c(\mathbf{x}) \in \mathscr{C} = \{1, 2, \ldots, C\}$, we define a prototype set of vectors $W = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_M\} \subseteq \mathbb{R}^n$ such that every $\mathbf{w} \in W$ has a corresponding class $c(\mathbf{w}) \in \mathscr{C}$. We divide the training set into the train and test sets, respectively. Using the train set along with standard LVQ training procedure, the learned prototypes $\mathbf{w}_k \in W$ together with its classes $c(\mathbf{w}_k)$ are accessed and applied in accordance with the fuzzy probabilistic assignments of FCM described in $(2.1a)$ to determine the classification label security of the test set. So for test data, the classification label security is calculated and returned accordingly.

We further consider the utilization of the computed classification label securities to determine reject classification and non-reject classification strategy [18]. Advancing in this regard, we consider a test sample $\mathbf{x}_k \subseteq \mathbb{R}^n$ for all $1 \leq k \leq N$, a given model classifier function indicated by M_c and the computed classification label security of $\mathbf{x}_k$ indicated by u_{ik}, $1 \leq i \leq |\mathscr{C}|$ and define a non-reject classification strategy based on

$$M_c(\mathbf{x}_k) = c_i \in \mathscr{C} \ \arg\max_i \ \{u_{ik}\} \tag{3.23}$$

and a reject classification strategy based on

$$M_c(\mathbf{x}_k) = \begin{cases} r, & if \ \ u_{ik} < h \quad \forall\, i \\ c_i \in \mathscr{C}, & \arg\max_i \ \{u_{ik}\} \quad \text{otherwise} \end{cases} \tag{3.24}$$

with an extended class set $\mathscr{C}^* = \mathscr{C} \cup \{r\}$ where the decision to reject is indicated by r based on a fixed but arbitrarily choosen threshold classification label security h, $0 \leq h \leq 1$ [18].

The average model classification certainty which indicates regions in the data space where the model is confident with respect to the prototypes is indicated by

$$\zeta(X, W) = \frac{1}{|W|} \sum_{\mathbf{w} \in W} (\zeta_{\mathbf{w}}(X))$$

where $\zeta_{\mathbf{w}}(X)$ in (3.25) measures the classification certainty of respective prototype $\mathbf{w}$ and class responsibilities $c(\mathbf{w})$ with regards to equation (3.24) [11]

$$\zeta_{\mathbf{w}}(X) = \frac{|\left\{\mathbf{x} \in X | \mathbf{w} = \mathbf{w}_{s(\mathbf{x})} \wedge c(\mathbf{x}) = c(\mathbf{w}_{s(\mathbf{x})})\right\}|}{|\left\{\mathbf{x} \in X | \mathbf{w} = \mathbf{w}_{s(\mathbf{x})}\right\}|} \tag{3.25}$$

The behavior of the models concerning the test accuracy Acc and the adjusted test accuracy Acc_h not including rejected classification based on a given threshold security

h will also be investigated with

$$Acc_h = \frac{|\{\mathbf{x} \in X |\ c(\mathbf{x}) = c(\mathbf{w}_{s(\mathbf{x})})\}|}{|X|} \tag{3.26}$$

for accuracy consideration disregarding any rejected classification, we drop the threshold security h. The model classification certainty $\zeta(X, W)$ will be utilized as the primary metric to evaluate the confidence of the GLVQ, GMLVQ and CELVQ models used in this thesis for the determination of the classification label security.

4 Experimental Results

4.1 General Overview of Train/Test Procedure

A standard and generally accepted procedure in machine learning for model training and testing involve demarcating the data set under consideration into a train set and test set. The ratio of the demarcation puts more weight on the train set than the test set. A good model should endure vigorous training with much of the data set in order to capture a reasonably representable variance per the patterns present in the data set with the remainder of a relatively sizeable unused data points tested on the model to evaluate how well the model can predict with new data points.

A significant way forward will be to have a fair explorable insight of the data points in the data set under study. This will lead to the decision on which data scaling procedure to apply to the data set. In this thesis, all data sets used in the experimentation were split into the train-test ratio of $4:1$. The data sets were normalized with (4.1) in the data preparation stage.

$$\mathbf{x}_s = \frac{\mathbf{x} - \text{mean}(\mathbf{x})}{\text{standard deviation}(\mathbf{x})} \tag{4.1}$$

$\mathbf{x}_s$ is the normalized vector and $\mathbf{x}$ is the unnormalized vector. The train set is first fitted and transformed with the standard feature scaler in (4.1) whilst the mean$(\mathbf{x})$ of the train set and the standard deviation$(\mathbf{x})$ of the train set is used to transform the test set. Generally, this is done to disallow information passage into the model during the testing stage. The split must be done to ensure that the training and test sets are mutually disjoint sets.

4.2 Iris Data Set

The Iris data set [19] is used in this thesis to determine the classification label security by taking into account the fuzzy probabilistic assignment of FCM estimates. This data set is chosen primarily to reflect its prolific usage for most machine learning implementation schemes. The Iris data set holds an unchallenged position of fame in the machine learning community and remains well understood in such regard. The data set has 150 data points present with three uniform classes, each containing 50 data points with four features, namely sepal length in cm, sepal width in cm, petal length in cm and petal width in cm. The three classes are referred to as Iris Setosa, Iris Versicolour and Iris Virginica.

4.3 Classification Label Security of Iris Data set

The Iris data set is normalized as described in (4.1) and in-line with standard train-test procedure, the train samples consists of 80% of the total data points, with the remaining 20% used as test samples. The prototypes initialization is done uniformly across all three classes with one prototype per class. Training is realized using batches with 32 samples for 100 maximum epochs with $\eta = 0.01$. The training of the Iris train set was realized using the python implementation [20]. The learned prototypes were accessed and used to determine the classification label security of the Iris test set. The GLVQ, GMLVQ and CELVQ models were employed in the learning and classification of the Iris data set. A summary of computed results that indicate the adjusted test accuracy with and without rejected classifications for the GLVQ, GMLVQ and CELVQ models is summarised in Table 4.3. The model classification certainty is summarised in Table 4.1 and 4.2.

Model classification certainty of the Iris test set				
Model	$\zeta_{\mathbf{w}}^{0}(X)$	$\zeta_{\mathbf{w}}^{1}(X)$	$\zeta_{\mathbf{w}}^{2}(X)$	$\zeta(X,W)$
GLVQ	1.00	0.69	0.89	0.860
GMLVQ	1.00	0.69	0.88	0.857
CELVQ	1.00	0.69	0.89	0.860

Table 4.1: This table contains a summary of the model classification certainty of the Iris test set with non-reject classification. $\zeta_{\mathbf{w}}^{0}(X)$, $\zeta_{\mathbf{w}}^{1}(X)$ and $\zeta_{\mathbf{w}}^{2}(X)$ indicates the classification certainty of the model prototypes with respect to the Iris Setosa, Iris Versicolour and Iris Virginica classes. The average model classification certainty for the Iris test set is indicated by $\zeta(X,W)$.

Model classification certainty of the Iris test set $(h = 0.7)$				
Model	$\zeta_{\mathbf{w}}^{0}(X)$	$\zeta_{\mathbf{w}}^{1}(X)$	$\zeta_{\mathbf{w}}^{2}(X)$	$\zeta(X,W)$
GLVQ	1.00	1.00	1.00	1.00
GMLVQ	1.00	0.69	1.00	0.90
CELVQ	1.00	1.00	1.00	1.00

Table 4.2: This table contains a summary of the model classification certainty of the Iris test set with reject classification based on a threshold classification label security of 0.7. $\zeta_{\mathbf{w}}^{0}(X)$, $\zeta_{\mathbf{w}}^{1}(X)$ and $\zeta_{\mathbf{w}}^{2}(X)$ indicates the classification certainty of the model prototypes with respect to the Iris Setosa, Iris Versicolour and Iris Virginica classes. The average model classification certainty for the Iris test set is indicated by $\zeta(X,W)$.

Model	Test Accuracy (Acc)	Adjusted Test Accuracy (Acc_h)
GLVQ	0.83	1.00
GMLVQ	0.83	0.84
CELVQ	0.83	1.00

Table 4.3: This table contains a summary of the model classification test accuracy of the Iris test set based on a non-reject classification (3.23) and a reject classification (3.24) based on a threshold classification label security $(h = 0.7)$.

By the estimates in Table 4.1, 4.2 and 4.3 we can infer how accurate the models are when they are confident regarding the predictions made. In other words, the certainty with which the models (GLVQ, GMLVQ and CELVQ) made the observed classifications from the Iris test set. We relate this to the computed classification label securities and observe by way of visualization (Figures 4.3, 4.6 and 4.9), regions in the Iris data space where the models (GLVQ, GMLVQ and CELVQ) are confident or unconfident about the classification labels. We further explore Figure 4.3 to Figure 4.9 where we can determine for any arbitrarily chosen threshold, regions in the data space where the models are confident or unconfident about the classification labels. The utilization of reject classification strategy (3.24) was able to improve the model classification certainty both at the class and overall level.

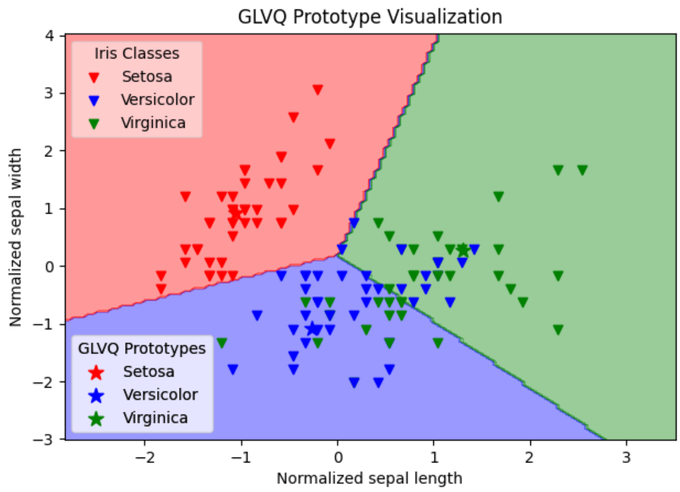

Figure 4.1: Iris train set with GLVQ prototypes and decision boundary

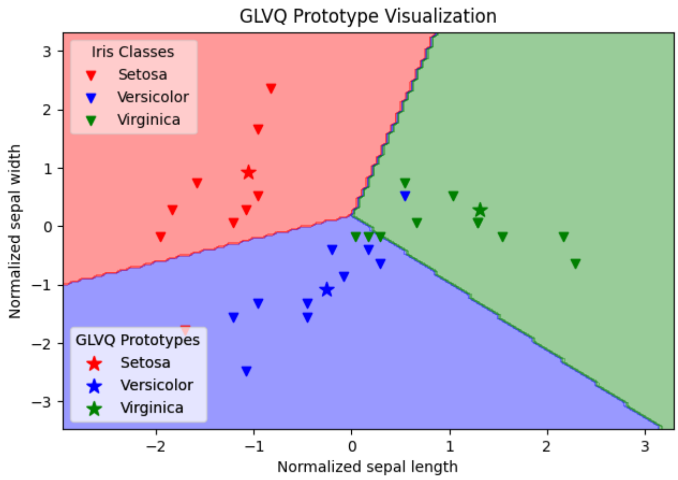

Figure 4.2: Iris test set with GLVQ prototypes and decision boundary

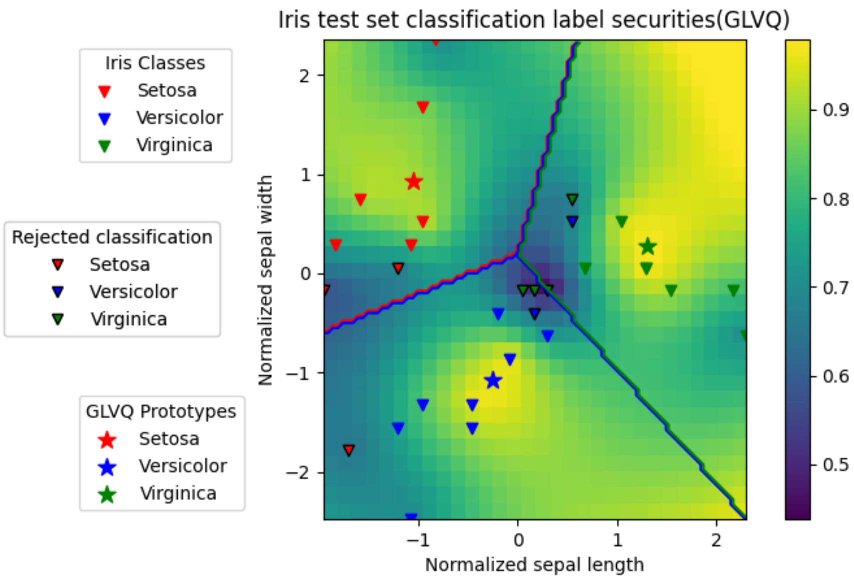

Figure 4.3: The data space of the Iris test set showing the GLVQ model computed classification label securities with a threshold security $(h = 0.7)$.

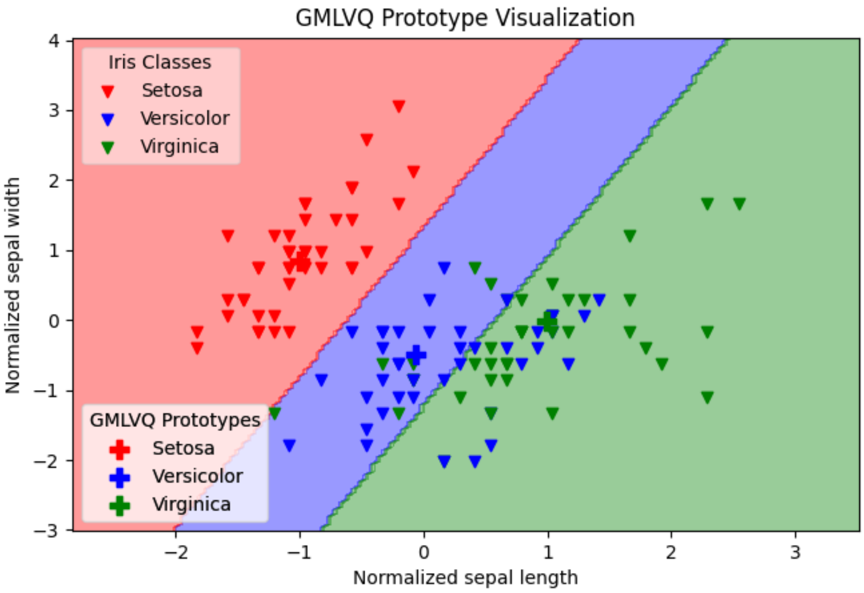

Figure 4.4: Iris train set with GMLVQ prototypes and decision boundary

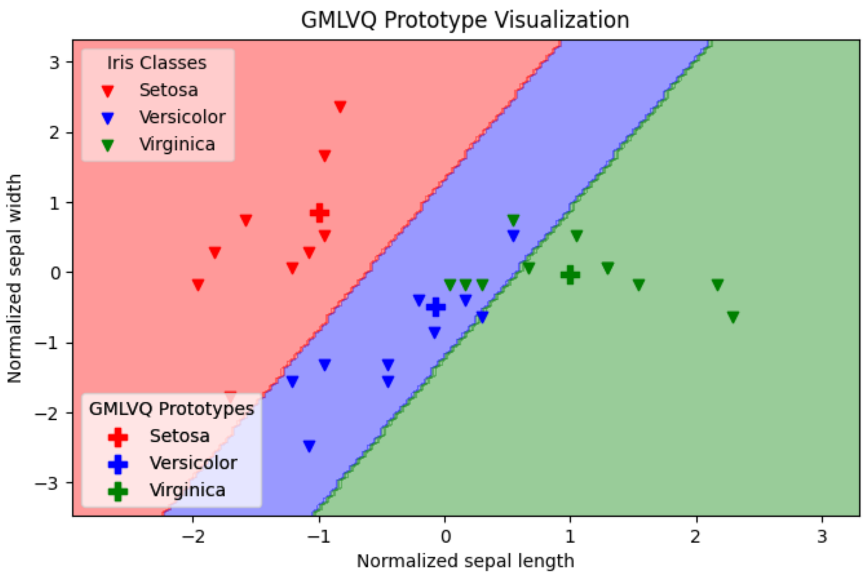

Figure 4.5: Iris test set with GMLVQ prototypes and decision boundary

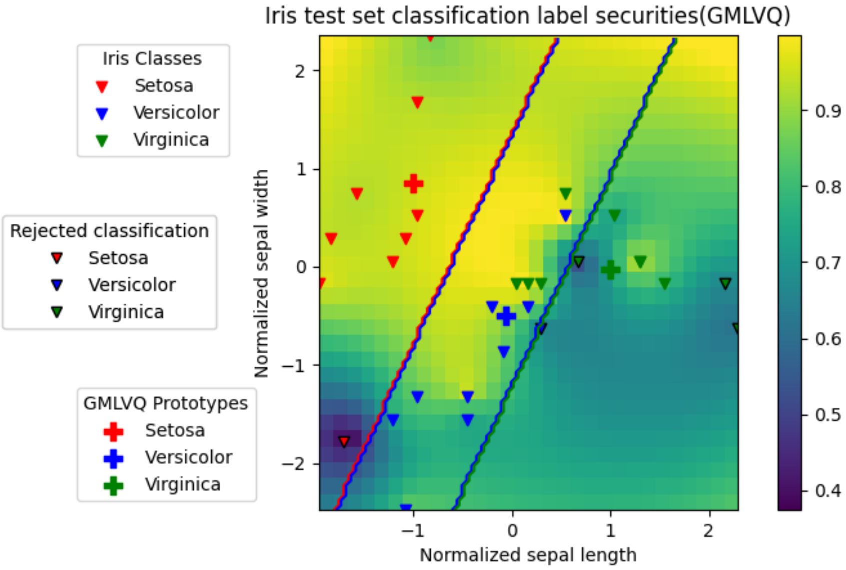

Figure 4.6: The data space of the Iris test set showing the GMLVQ model computed label securities with a threshold security $(h = 0.7)$.

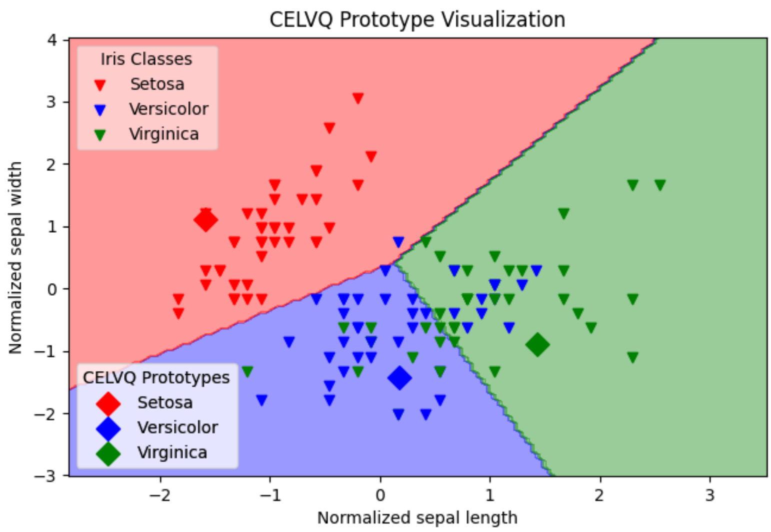

Figure 4.7: Iris train set with CELVQ prototypes and decision boundary

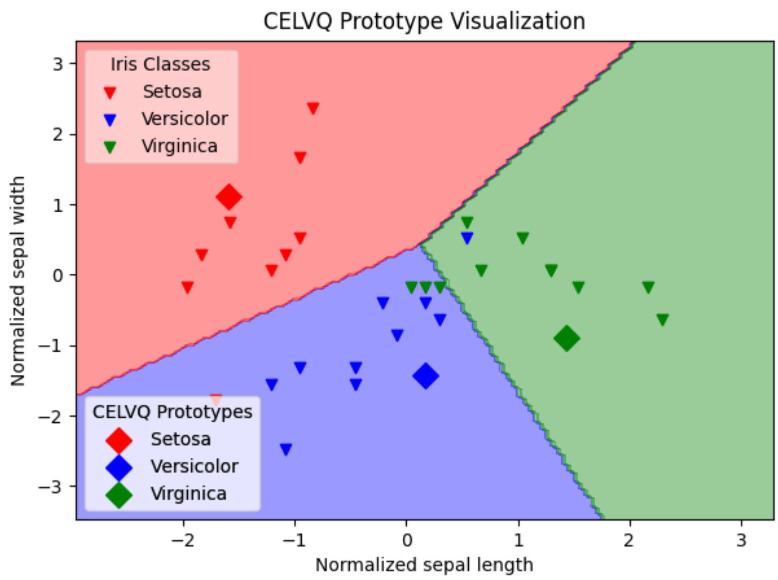

Figure 4.8: Iris test set with CELVQ prototypes and decision boundary

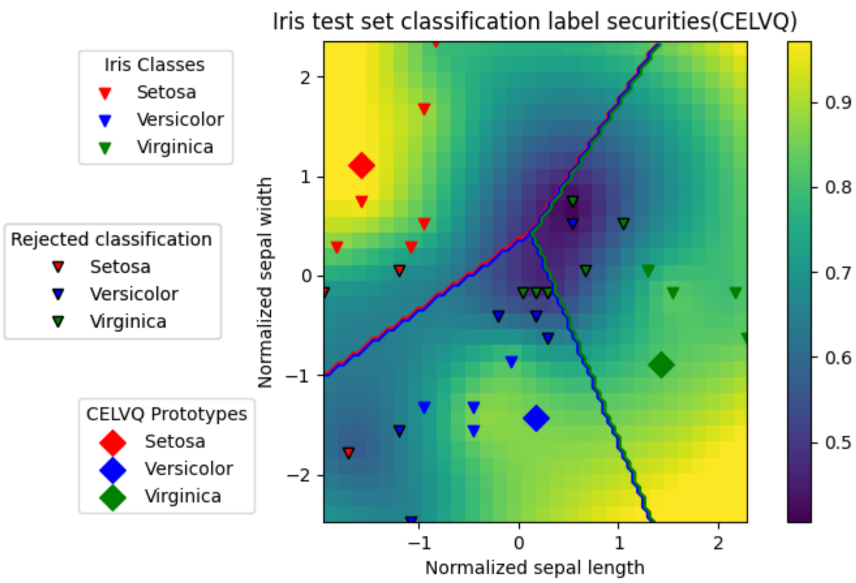

Figure 4.9: The data space of the Iris test set showing the CELVQ model computed classification label securities with a threshold security $(h = 0.7)$.

From the results in Tables 4.1 and 4.2, we observe for all the three models (GLVQ, GMLVQ and CELVQ) that, the model classification certainty with and without rejected classifications for the Iris Setosa class was 1.0. By referring to Figures 4.3, 4.6 and 4.9, we can determine for sure whether this recorded certainty and the level of confidence in the regions of the Iris Setosa class labels were in agreement or not. So for an arbitrarily chosen label security threshold of 0.7, we observe that most of the classification labels in the region for the Iris Setosa class had very high label securities. This analogy applies to the observed certainties of the Iris Versicolour and Iris Virginica class as well. The utilization of reject classification strategy (3.24) was able to improve the test accuracy for all the models. By this implementation, we have determined the extent to which the predicted labels of the Iris test can be trusted.

4.4 Breast Cancer Wisconsin (Diagnostic) Data set (WDBC)

This thesis proceeds to test the implementation of the determination of the classification label security on the well-acclaimed WDBC data set [21], encompassing 569 data points along with 30 numeric, predictive attributes (specified by the mean, standard error and worst). So for each data point, measurements are made under the attributes designations: radius, texture, perimeter, area, smoothness, compactness, concactivity, concave points, symmetry and fractal dimension. Two classes, namely WDBC-Malignant and WDBC-Benign, are primarily considered with somewhat relatively homogeneous class distributions in the ratio of 212:357 for Malignant and Benign classes, respectively.

4.5 Classification Label Security of Breast Cancer Wisconsin(Diagnostic) Data set

All standard procedure described in section 4.3 for the training with Iris data set is maintained and employed to train the WDBC data set. Considering a train-test split of $80\% : 20\%$ for the WDBC data set, the prototypes initialization was done uniformly across the classes with one prototype per class. Training is realized using batches with 32 samples for 100 maximum epochs with $\eta = 0.01$. The learned prototypes from the WDBC trained data using the GLVQ, GMLVQ and CELVQ models were accessed and used to determine the classification label security of the WDBC test set. A summary of computed results that indicate the adjusted test accuracy without rejected classifications for the GLVQ, GMLVQ and CELVQ models is summarized in Table 4.6.

Model classification certainty of the WDBC test set			
Model	$\zeta_{\mathbf{w}}^{0}(X)$	$\zeta_{\mathbf{w}}^{1}(X)$	$\zeta(X,W)$
GLVQ	0.89	0.86	0.875
GMLVQ	0.90	0.91	0.905
CELVQ	0.88	0.91	0.895

Table 4.4: This table contains a summary of the model classification certainty of the WDBC test set with non-reject classification. $\zeta_{\mathbf{w}}^{0}(X)$ and $\zeta_{\mathbf{w}}^{1}(X)$ indicates the classification certainty of the model prototypes with respect to the WDBC-Malignant and WDBC-Benign classes. The average model classification certainty for the WDBC test set is indicated by $\zeta(X,W)$.

Model classification certainty of the WDBC test set $(h = 0.7)$			
Model	$\zeta_{\mathbf{w}}^{0}(X)$	$\zeta_{\mathbf{w}}^{1}(X)$	$\zeta(X,W)$
GLVQ	1.00	0.92	0.960
GMLVQ	0.93	0.94	0.935
CELVQ	1.00	1.00	1.000

Table 4.5: This table contains a summary of the model classification certainty of the WDBC test set with reject classification based on a threshold classification label security of 0.7. $\zeta_{\mathbf{w}}^{0}(X)$ and $\zeta_{\mathbf{w}}^{1}(X)$ indicates the classification certainty of the model prototypes with respect to the WDBC-Malignant and WDBC-Benign classes. The average model classification certainty for the WDBC test set is indicated by $\zeta(X,W)$.

Model	Test Accuracy (Acc)	Adjusted Test Accuracy (Acc_h)
GLVQ	0.87	0.94
GMLVQ	0.90	0.94
CELVQ	0.89	1.00

Table 4.6: This table contains a summary of the model classification test accuracy of the WDBC test set based on a non-reject classification (3.23) and a reject classification (3.24) based on a threshold classification label security $(h = 0.7)$.

Observing estimates in Table 4.4, 4.5 and 4.6, we can draw an inference on how accurate the models are when they are confident regarding the classifications labels of the test set. In other words, the certainty with which the models (GLVQ, GMLVQ and CELVQ) made the observed classifications from the WDBC test set. We relate this to the computed classification label securities and show by way of visualization (Figures 4.12, 4.15, 4.18), regions in the WDBC data space where the models (GLVQ, GMLVQ and CELVQ) are confident or unconfident about the classification labels. We further explore Figure 4.12 to Figure 4.18 where we can determine for any arbitrarily chosen threshold, regions in the WDBC data space where the models are confident or unconfident about the classification labels.

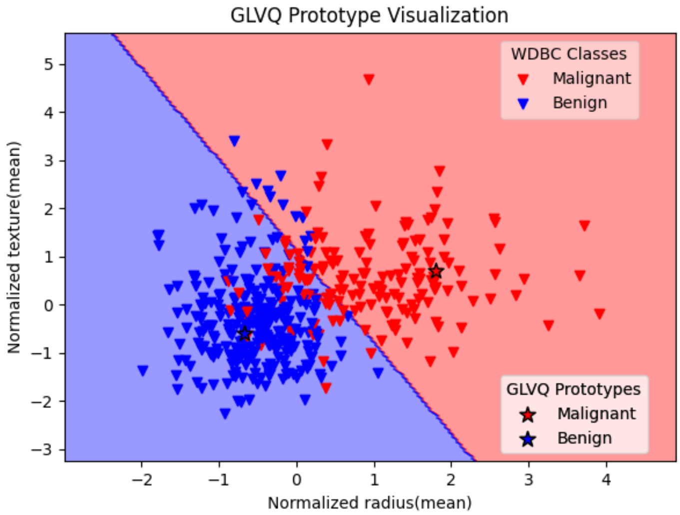

Figure 4.10: WDBC train set with GLVQ prototypes and decision boundary

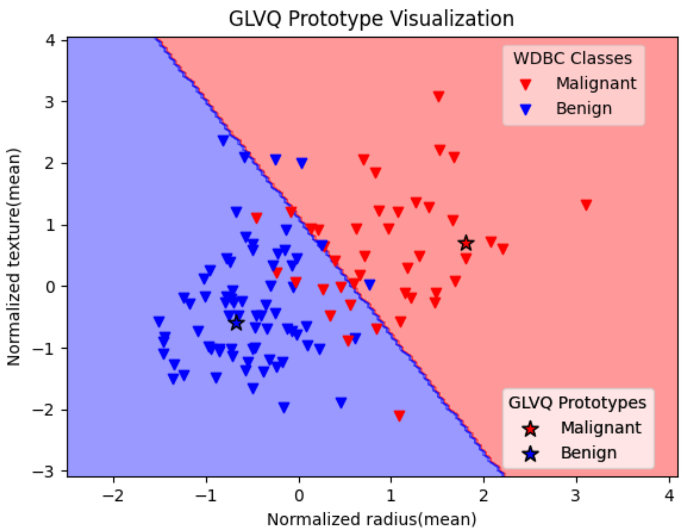

Figure 4.11: WDBC test set with GLVQ prototypes and decision boundary

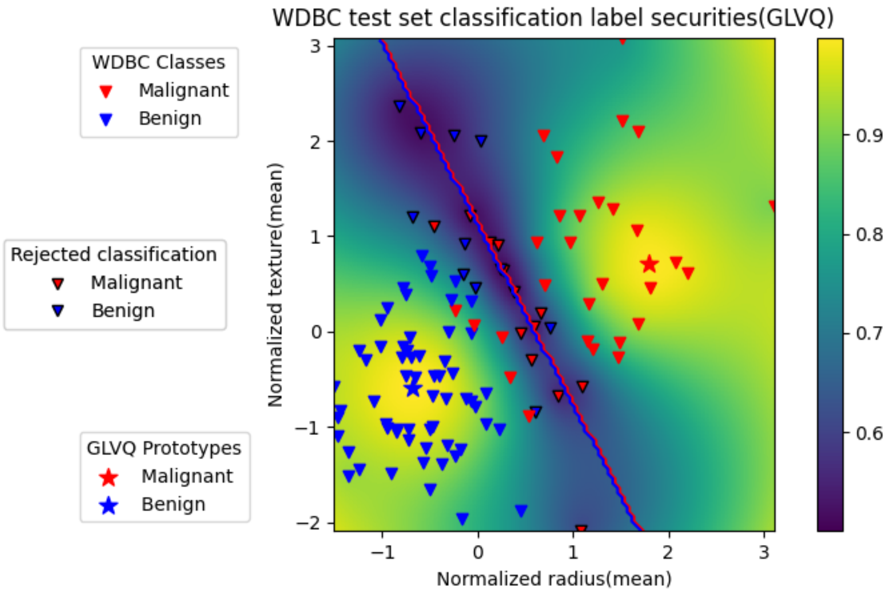

Figure 4.12: The data space of the WDBC test set showing the GLVQ model computed classification label securities with a threshold security $(h = 0.7)$.

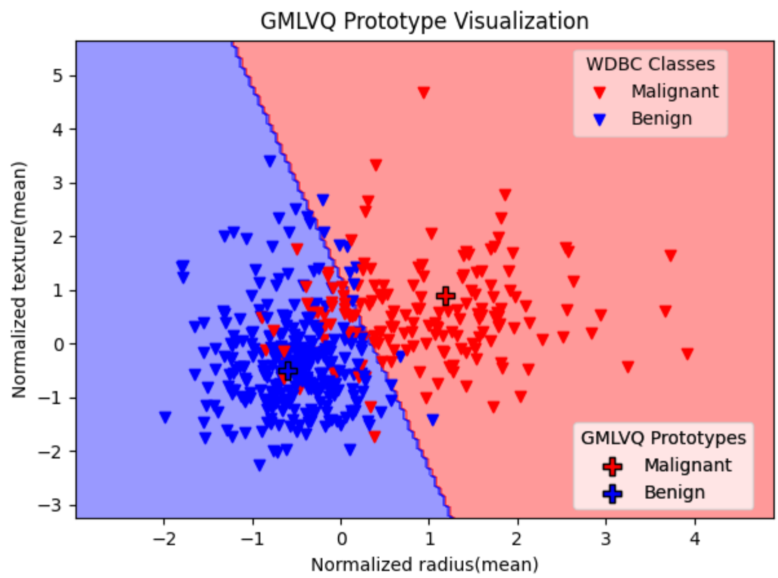

Figure 4.13: WDBC train set with GMLVQ prototypes and decision boundary

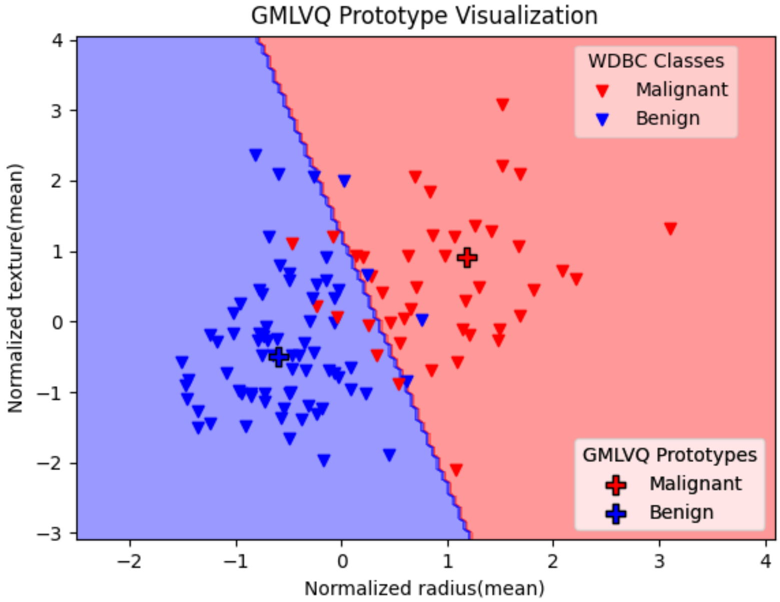

Figure 4.14: WDBC test set with GMLVQ prototypes and decision boundary

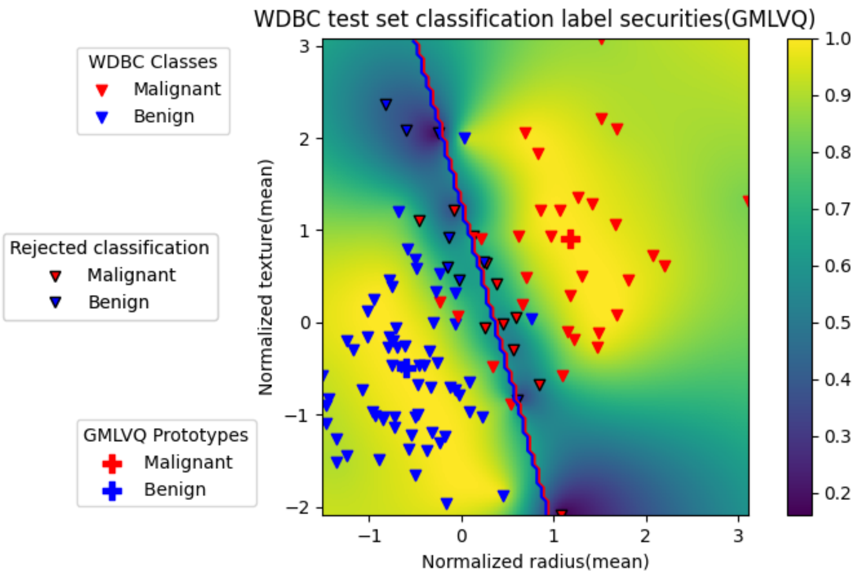

Figure 4.15: The data space of the WDBC test set showing the GMLVQ model computed classification label securities with a threshold security $(h = 0.7)$.

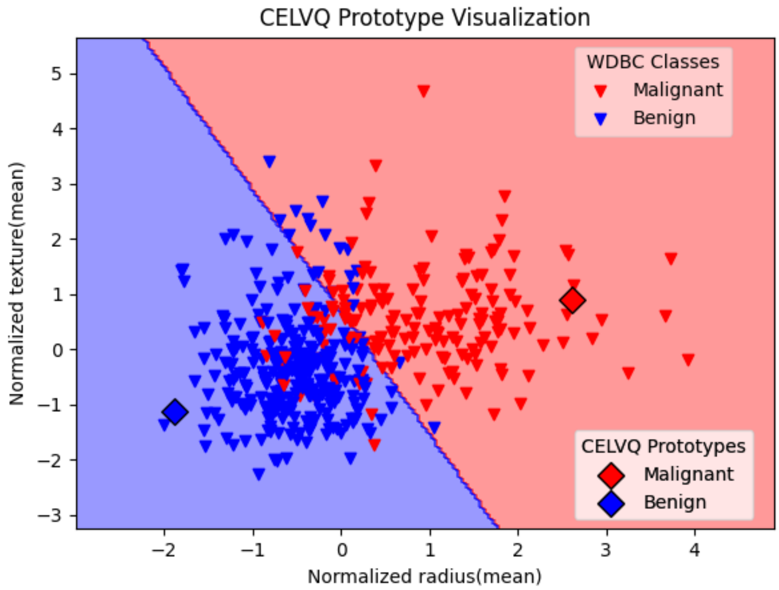

Figure 4.16: WDBC train set with CELVQ prototypes and decision boundary

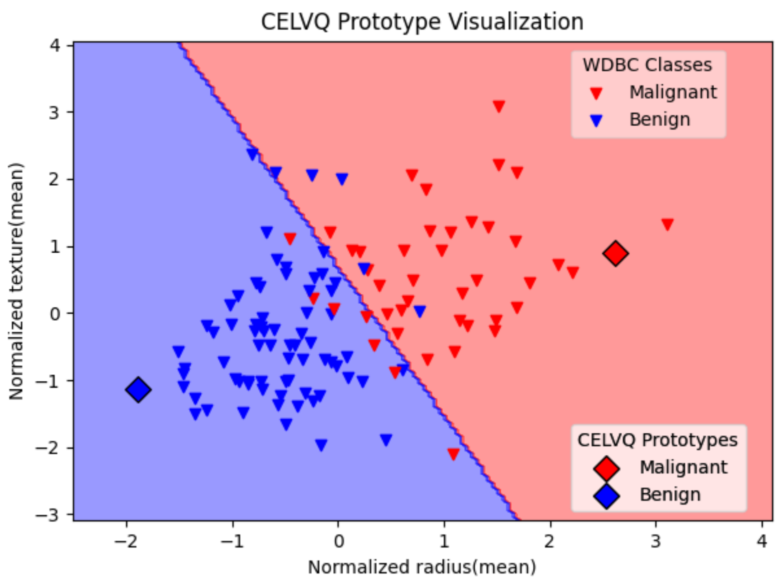

Figure 4.17: WDBC test set with CELVQ prototypes and decision boundary

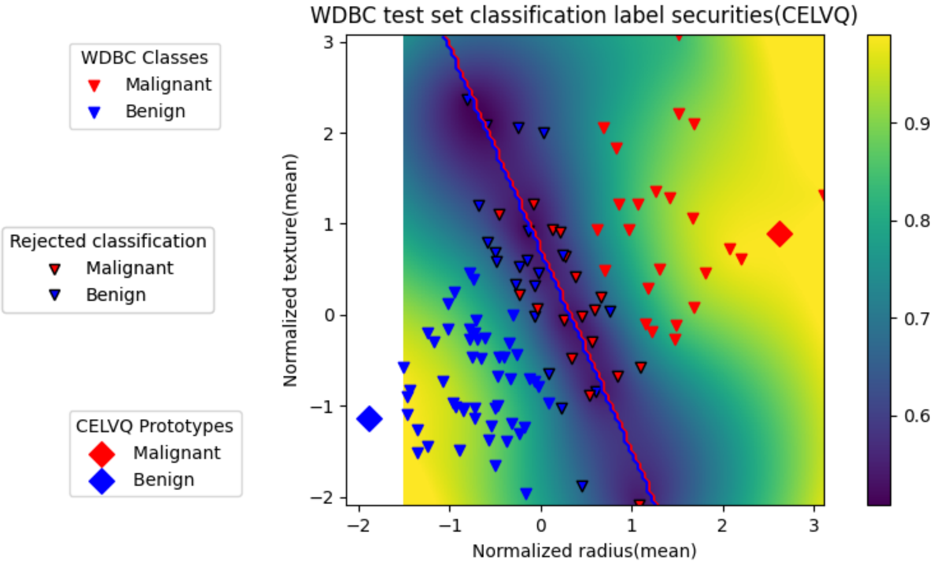

Figure 4.18: The data space of the WDBC test set showing the CELVQ model computed label securities with a threshold security $(h = 0.7)$.

Similarly, from the results in tables 4.4 and 4.5, we observe for all the three models (GLVQ, GMLVQ and CELVQ), the model classification certainty with and without rejected classifications for the WDBC test set. By referring to Figures 4.12, 4.15 and 4.18, we can determine for sure whether this recorded certainty and the levels of confidence in the regions of the WDBC- Malignant and Benign class labels were in agreement or not. So for an arbitrary chosen label security threshold of 0.7, we observe improvements in the model classification certainty for both classes of the WDBC-test set. This same deduction applies to the average model classification certainty. The adjusted test accuracy, not including rejected classification, indicated improvements in the test accuracy, which gives insight into how the models were accurate about their determined confidence. By this implementation, we have determined the extent to which the predicted labels of the WDBC test set can be trusted.

5 Conclusion and Prospective Work

This thesis investigated to determine the classification label security using fuzzy probabilistic assignments of FCM estimates. Chapter 4 exhibited by implementation, computation of the classification label security for the GLVQ, GMLVQ and CELVQ models. So for a given test set, the classification label security of all predicted labels is computed. The visualization of this implementation was accompanied by displaying the regions in the data space for which the considered models were confident or unconfident regarding the classification labels for the data sets used in the experimentations. We also determined the accuracy to which the models made the classifications when they were confident. The classification label security in this regard has been determined. Concerning future work, a possibilistic approach to the determination of classification label security will be considered.

Appendix A: Reference Implementation in Python

A.1: label_security1.py

```python
"""Module to Determine classification Label Security/Certainty"""
import numpy as np
from scipy.spatial import distance


class LabelSecurity:
    """
    Label Security
    :params

    x_test: array, shape=[num_data,num_features]
    Where num_data is the number of samples and num_features
      refers to the number of features.

    class_labels: array-like, shape=[num_classes]
    Class labels of prototypes

    predict_results:  array-like, shape=[num_data]
    Predicted labels of the test-set

    model_prototypes: array-like, shape=[num_prototypes, num_features]

    Prototypes from the trained model using train-set, where
      num_prototypes refers to the number of prototypes

    x_dat : array, shape=[num_data, num_features]
      Input data

    fuzziness_parameter: int, optional(default=2)
    """

    def __init__(self, x_test, class_labels, predict_results,
        model_prototypes, x_dat, fuzziness_parameter=2):
        self.x_test = x_test
        self.class_labels = class_labels
```

```python
        self.predict_results = predict_results
        self.model_prototypes = model_prototypes
        self.x_dat = x_dat
        self.fuzziness_parameter = fuzziness_parameter

    def label_sec_f(self, x):
        """
        Computes the labels security of each prediction from
        the model using the test_set

        :param x
        predicted labels from the model using test-set
        :return:
        labels with their security
        """

        security = float

        # Empty list to populate with certainty of labels
        my_label_sec_list = []

        # loop through the test_set
        for i in range(self.x_test.shape[0]):
        # consider respective class labels
            for label in range(self.class_labels.shape[0]):
        # checks where the predicted label equals the class label
                if self.predict_results[i] == label:
        # computes the certainty/security per predicted label
                    ed_dis = distance.euclidean(self.x_test[i,
                                  0:self.x_dat.shape[1]],
                                  self.model_prototypes[label,
                                  0:self.x_dat.shape[1]])
                    sum_dis = 0
                    for j in range(
                      self.model_prototypes.shape[0]):
                        sum_dis += np.power(
                            ed_dis /
                            distance.euclidean(self.x_test[i,
                                      0:self.x_dat.shape[1]],
                                      self.model_prototypes[j,
                                      0:self.x_dat.shape[1]]),
                            2 / (self.fuzziness_parameter - 1))
```

```
                        )
                        security = 1 / sum_dis

        my_label_sec_list.append(np.round(security, 4))  # add the computed label certainty to list above
        my_label_sec_list = np.array(my_label_sec_list)
        my_label_sec_list = my_label_sec_list.reshape(len(
            my_label_sec_list), 1)  # reshape list to 1-D array
        x = np.array(x)
        x = x.reshape(len(x), 1)  # reshape predicted labels into 1-D array
        labels_with_certainty = np.concatenate(
          (x, my_label_sec_list), axis=1)
        return labels_with_certainty


class LabelSecurityM:
    """
    label security for matrix GLVQ
    :parameters

    x_test: array, shape=[num_data, num_features]
    Where num_data refers to the number of samples and
        num_features refers to the number of features

    class_labels: array-like, shape=[num_classes]
    Class labels of prototypes

    model_prototypes: array-like, shape=[num_prototypes,
        num_features]

    Prototypes from the trained model using train-set, where
        num_prototypes refers to the number of prototypes

    model_omega: array-like, shape=[dim, num_features]
    Omega_matrix from the trained model, where dim is an int
        refers to the maximum rank

    x:  array, shape=[num_data, num_features]
    Input data

    fuzziness_parameter=int, optional(default=2)
```

```
    """

    def __init__(self, x_test, class_labels, model_prototypes,
        model_omega, x, fuzziness_parameter=2):
        self.x_test = x_test
        self.class_labels = class_labels
        self.model_prototypes = model_prototypes
        self.model_omega = model_omega
        self.x = x
        self.fuzziness_parameter = fuzziness_parameter

    def label_security_m_f(self, x):
        """
        Computes the label security of each prediction from
        the model using the test_set
        :param x: predicted labels from the model using X_test
        :return: labels with their securities
        """
        security = " "
        # Empty list to populate with the label certainty
        my_label_security_list = []
        # loop through the test set
        for i in range(len(self.x_test)):
        # considers respective class labels of prototypes
            for label in range(len(self.class_labels)):
        # checks if predicted label equals class label of prototypes
                if x[i] == label:

        # computes the label certainty per predicted label
                    standard_ed = (self.x_test[i,
                                0:self.x.shape[1]] -
                                self.model_prototypes[label,
                                0:self.x.shape[1]])
                    squared_ed = standard_ed.T.dot(
                        self.model_omega.T).dot(
                        self.model_omega).dot(standard_ed)
                    sum_dis = 0
                    for j in range(
                      len(self.model_prototypes)):
                        standard_ed1 = (self.x_test[i,
                                0:self.x.shape[1]] -
                                self.model_prototypes[j,
```

```
                                        0:self.x.shape[1]])
                                sum_dis += np.power(
                                    squared_ed / (standard_ed1.T.dot(
                                        self.model_omega.T).dot(
                                        self.model_omega).dot(
                                        standard_ed1)),
                                    1 / (self.fuzziness_parameter - 1)
                                )
                            security = 1 / sum_dis

            # adds the computed certainty to the list
            my_label_security_list.append(np.round(security, 4))
            my_label_security_list = np.array(
              my_label_security_list)
            my_label_security_list = my_label_security_list\
              .reshape(len(my_label_security_list), 1)  # 1-D array reshape
            x = np.array(x)
            x = x.reshape(len(x), 1)  # reshape the predicted labels into 1-D array
            labels_with_certainty = np.concatenate(
              (x, my_label_security_list), axis=1)
            return labels_with_certainty


class LabelSecurityLM:
    """
    label security for local matrix GLVQ
    :parameters

    x_test: array, shape=[num_data, num_features]
    Where num_data refers to the number of samples and
      num_features refers to the number of features

    class_labels: array-like, shape=[num_classes]
    Class labels of prototypes

    model_prototypes: array-like, shape=[num_prototypes,
      num_features]

    Prototypes from the trained model using train-set, where
      num_prototypes refers to the number of prototypes

```

```
    model_omega: array-like, shape=[dim, num_features]
    Omega_matrix from the trained model, where dim is an int
      refers to the maximum rank

    x: array, shape=[num_data, num_features]
    Input data

    fuzziness_parameter=int, optional(default=2)
    """

    def __init__(self, x_test, class_labels, model_prototypes,
        model_omega, x, fuzziness_parameter=2):
        self.x_test = x_test
        self.class_labels = class_labels
        self.model_prototypes = model_prototypes
        self.model_omega = model_omega
        self.x = x
        self.fuzziness_parameter = fuzziness_parameter

    def label_security_lm_f(self, x):
        """
        computes the label security of each prediction from
        the model using the test_set
        and returns only labels their corresponding security.
        :param x: predicted labels from the model using X_test
        :return: labels with  security
        """
        security = " "
        # Empty list to populate with the label security
        my_label_security_list = []
        # loop through the test set
        for i in range(len(self.x_test)):
        # considers respective class labels of prototypes
            for label in range(len(self.class_labels)):
        # checks if predicted label equals class label of
           prototypes
                if x[i] == label:

        # computes the label certainty per predicted label
                    standard_ed = (self.x_test[i,
                                0:self.x.shape[1]] -
                                self.model_prototypes[label,
                                0:self.x.shape[1]])
```

```
                    squared_ed = standard_ed.T.dot(
                        self.model_omega[label].T).dot(
                        self.model_omega[label]).dot(
                        standard_ed)
                    sum_dis = 0
                    for j in range(len(
                        self.model_prototypes)):
                        standard_ed1 = (self.x_test[i,
                                0:self.x.shape[1]] -
                                self.model_prototypes[j,
                                0:self.x.shape[1]])
                        sum_dis += np.power(
                            squared_ed / (standard_ed1.T.dot(
                                self.model_omega[j].T).dot(
                                self.model_omega[j]).dot(
                                standard_ed1)),
                            1 / (self.fuzziness_parameter - 1)
                        )
                        security = 1 / sum_dis

        # adds the computed certainty to the list
        my_label_security_list.append(np.round(security, 4))
        my_label_security_list = np.array(
          my_label_security_list)
        my_label_security_list = my_label_security_list\
          .reshape(len(my_label_security_list), 1)  # 1-D array reshape
        x = np.array(x)
        x = x.reshape(len(x), 1)  # reshape the predicted labels into 1-D array
        labels_with_certainty = np.concatenate(
          (x, my_label_security_list), axis=1)
        return labels_with_certainty


if __name__ == '__main__':
print('import module to use')
```

A.2: contour.py

```python
"""
visualize the classification label securities
"""
import scipy.interpolate
import torch
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.lines import Line2D
from matplotlib.colors import ListedColormap

matplotlib.style.use('default')
class Contourrn:
    """
    visualize the classification label securities
    """
    def __init__(self):
        pass


    def plot_dec_boundary(self,
        x,
        y,
        model,
        model_p,
        title,
        xlabel,
        ylabel,
        model_type,
        model_index):

        """

        :param x: X_test
        :param y: labels of the test set
        :param model: model object
        :param model_p: model prototypes
        :param title: Title of plot
        :param xlabel: Title of data dimension 1
        :param ylabel: Title of data dimension 2
        :param model_type: string: Name of model
        :param model_index: int: model index
```

```
        :return:
        """

        colors = ["r", "b", "g", "y", "m"]
        colors_ = ["r", "b", "g"]
        marker = ["*", "P", "D", "p", "H"]
        cm = ListedColormap(colors_)
        ax = plt.gca()
        z1 = model_p
        # Plotting decision regions
        x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
        y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
        x1, y1 = np.meshgrid(np.arange(x_min, x_max, 0.05),
            np.arange(y_min, y_max, 0.05))

        y_pred_1 = model.predict(torch.Tensor(
            np.c_[x1.ravel(), y1.ravel()]))
        Z1 = y_pred_1.reshape(x1.shape)
        plt.contourf(x1, y1, Z1, alpha=0.4, cmap=cm)
        # customize the lines
        # for t in cont.collections:
        #     t.set_edgecolor('face')
        #     t.set_linewidth(0)
        # plt.savefig('test')

        for t in range(len(y)):
            if y[t] == 0:
                s1 = ax.scatter(
                    x[t, 0],
                    x[t, 1],
                    c='r',
                    marker='v',)
            if y[t] == 1:
                    s2 = ax.scatter(
                    x[t, 0],
                    x[t, 1],
                    c='b',
                    marker='v',)
            if y[t] == 2:
                    s3 = ax.scatter(
                    x[t, 0],
                    x[t, 1],
                    c='g',
```

```
                  marker='v',)
        legend1 = plt.legend((s1, s2, s3),
            ["Setosa", "Versicolor", "Virginica"],
            title="Iris Classes",
            loc="upper left",
            fancybox=True,
            framealpha=0.5)
        ax.add_artist(legend1)

        # plot learned prototypes
        t1 = ax.scatter(
            z1[0][0],
            z1[0][1],
            s=100,
            color=colors[0],
            marker=marker[model_index])
            t2 = ax.scatter(
            z1[1][0],
            z1[1][1],
            s=100,
            color=colors[1],
            marker=marker[model_index])
            t3 = ax.scatter(
            z1[2][0],
            z1[2][1],
            s=100,
            color=colors[2],
            marker=marker[model_index])
        legend2 = plt.legend(
            (t1, t2, t3),
            ["Setosa ", "Versicolor ","Virginica"],
            title=f"{model_type} Prototypes",
            loc="lower left",
            fancybox=True, framealpha=0.5)
        ax.add_artist(legend2)

        plt.title(title)
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)

        return plt.show()
```

```
    def plot__newt(self,
        x,
        y,
        label_sec,
        model_p,
        index_list,
        xlabel,
        ylabel,
        title ,
        model_1,
        model_type,
        model_index,
        h):
        """ visualize classification label securities per
        class for a test set including learned prototypes
        responsible for the classifications.

        :param x: X_test
        :param y: labels of the test set
        :param label_sec: List containing label securities
        :param model_p: model prototypes
        :param index_list: List containing(index of data point
        ,label, label security)
        :param title: Title of Plot
        :param ylabel: Title of data dimension 2
        :param xlabel: Title of data dimension 1
        :param model_1: model understudy
        :param model_type:string : model name
        :param model_index: int : 0 for glvq, 1 for gmlvq and
         2 for celvq
        :param h: threshold security
        :return: Plot
        """

        ax = plt.gca()
        k = []
        k1 = []
        colors = ["r", "b", "g", "y", "m"]
        marker = ["*", "P", "D", "p", "H"]
        z_ = label_sec
        z1 = model_p
        for j in index_list:
        k.append(x[j[0], 0])
```

```
        k1.append(x[j[0], 1])
        x1, y1 = np.linspace(
            np.min(k), np.max(k),len(k)), np.linspace(
            np.min(k1), np.max(k1), len(k1))
            x1, y1 = np.meshgrid(x1, y1)

        rbf = scipy.interpolate.Rbf(k, k1, z_,
            function='linear')
        zi = rbf(x1, y1)

        x_min, x_max = np.min(k), np.max(k)
        y_min, y_max = np.min(k1), np.max(k1)
        x11, y11 = np.meshgrid(np.arange(x_min, x_max, 0.05),
        np.arange(y_min, y_max, 0.05))
        y_pred_1 = model_1.predict(
            torch.Tensor(np.c_[x11.ravel(), y11.ravel()]))

        Z1 = y_pred_1.reshape(x11.shape)

        plt.contour(x11, y11, Z1, levels=3,
        colors = np.array([colors[0], colors[1],
            colors[1],colors[2]]))

        # plot the label securities regions
        plt.imshow(zi,
            vmin=np.min(z_),
            vmax=np.max(z_),
            origin='lower',
            extent=[np.min(k),
            np.max(k),
            np.min(k1),
            np.max(k1)])

        # plot classification label securities together with
          rejected classifications based on a threshold
          security
        j = -1
        for j1 in index_list:
            j += 1
            if y[j] == 0 and j1[1] == 0 and j1[2] >= h:
                s1 = ax.scatter(
                    x[j, 0],
                    x[j, 1],
```

```
                    color='r',
                    marker='v')
            if y[j] == 0 and j1[1] != 0 and j1[2] >= h:
                s1_ = ax.scatter(
                    x[j, 0],
                    x[j, 1],
                    color='r',
                    marker='v')
            if y[j] == 0 and j1[1] == 0 and j1[2] < h:
                s1__ = ax.scatter(
                    x[j, 0],
                    x[j, 1],
                    color='r',
                    marker='v',
                    edgecolor='k')
            if y[j] == 0 and j1[1] != 0 and j1[2] < h:
                s1_ = ax.scatter(
                    x[j, 0],
                    x[j, 1],
                    color='r',
                    marker='v',
                    edgecolor='k')

            if y[j] == 1 and j1[1] == 1 and j1[2] >= h:
                s2 = ax.scatter(
                    x[j, 0],
                    x[j, 1],
                    color='b',
                    marker='v')
            if y[j] == 1 and j1[1] != 1 and j1[2] >= h:
                s2_ = ax.scatter(
                    x[j, 0],
                    x[j, 1],
                    color='b',
                    marker='v')
            if y[j] == 1 and j1[1] == 1 and j1[2] < h:
                s2__ = ax.scatter(
                    x[j, 0],
                    x[j, 1],
                    color='b',
                    marker='v',
                    edgecolor='k')
            if y[j] == 1 and j1[1] != 1 and j1[2] < h:
```

```python
                s2_ = ax.scatter(
                    x[j, 0],
                    x[j, 1],
                    color='b',
                    marker='v',
                    edgecolor='k')

            if y[j] == 2 and j1[1] == 2 and j1[2] >= h:
                s3 = ax.scatter(
                    x[j, 0],
                    x[j, 1],
                    color='g',
                    marker='v')
            if y[j] == 2 and j1[1] != 2 and j1[2] >= h:
                s3_ = ax.scatter(
                    x[j, 0],
                    x[j, 1],
                    color='g',
                    marker='v')
            if y[j] == 2 and j1[1] == 2 and j1[2] < h:
                s3__ = ax.scatter(
                    x[j, 0],
                    x[j, 1],
                    color='g',
                    marker='v',
                    edgecolor='k')
            if y[j] == 2 and j1[1] != 2 and j1[2] < h:
                s3_ = ax.scatter(
                    x[j, 0],
                    x[j, 1],
                    color='g',
                    marker='v',
                    edgecolor='k')

        legend1 = plt.legend((s1, s2, s3,),
            ["Setosa", "Versicolor", "Virginica"],
            title="Iris Classes",
            loc="upper left", bbox_to_anchor=(-0.6, 1))
        ax.add_artist(legend1)

        # plot the learned prototypes
        t1 = ax.scatter(z1[0][0],
            z1[0][1],
```

```
            s=100,
            color=colors[0],
            marker=marker[model_index])
        t2 = ax.scatter(z1[1][0],
            z1[1][1], s=100,
            color=colors[1],
            marker=marker[model_index])
        t3 = ax.scatter(z1[2][0],
            z1[2][1],
            s=100,
            color=colors[2],
            marker=marker[model_index])
        legend2 = plt.legend((t1, t2, t3,),
            ["Setosa ", "Versicolor ", "Virginica"],
            title=f"{model_type} Prototypes", loc="lower left",
            bbox_to_anchor=(-0.6, 0.0))
        ax.add_artist(legend2)

        legend_list = []
        for class_, color in zip(
          ["Setosa ", "Versicolor ", "Virginica"],
          ['r', 'b', 'g']):
            legend_list.append(Line2D([0], [0],
            marker='v', label=class_, ls='None',
            markerfacecolor=color,
            markeredgecolor='k'))
        legend3 = plt.legend(
            handles=legend_list,
            loc="center", bbox_to_anchor=(-0.5, 0.5),
            title='Rejected classification')
        ax.add_artist(legend3)

        plt.colorbar()
        plt.title(title)
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)

        return plt.show()

if __name__ == '__main__':
print('import module to use')
```

Bibliography

[1] J. C. Bezdek, “Pattern recognition with fuzzy objective function algorithms,” 1981.

[2] T. Kohonen, “Learning vector quantization,” in *Self-organizing maps*. Springer, 2001, pp. 245–261.

[3] A. Sato and K. Yamada, “Generalized learning vector quantization,” *Advances in neural information processing systems*, pp. 423–429, 1996.

[4] M. Biehl, A. Ghosh, and B. Hammer, “Learning vector quantization: The dynamics of winner-takes-all algorithms,” *Neurocomputing*, vol. 69, no. 7-9, pp. 660–670, 2006.

[5] M. Biehl, B. Hammer, and P. Schneider, “Matrix learning in learning vector quantization,” *IFL Technical Report Series*, 01 2006.

[6] B. Hammer, M. Strickert, and T. Villmann, “On the generalization ability of grlvq networks,” *Neural Processing Letters*, vol. 21, no. 2, pp. 109–120, 2005.

[7] B. Hammer and T. Villmann, “Generalized relevance learning vector quantization,” *Neural Networks*, vol. 15, no. 8-9, pp. 1059–1068, 2002.

[8] P. Schneider, M. Biehl, and B. Hammer, “Adaptive relevance matrices in learning vector quantization,” *Neural computation*, vol. 21, no. 12, pp. 3532–3561, 2009.

[9] K. Bunte, P. Schneider, B. Hammer, F.-M. Schleif, T. Villmann, and M. Biehl, “Limited rank matrix learning, discriminative dimension reduction and visualization,” *Neural Networks*, vol. 26, pp. 159–173, 2012.

[10] I. Gath and A. B. Geva, “Unsupervised optimal fuzzy clustering,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 11, no. 7, pp. 773–780, 1989.

[11] A. Villmann, M. Kaden, S. Saralajew, and T. Villmann, “Probabilistic learning vector quantization with cross-entropy for probabilistic class assignments in classification learning,” in *International Conference on Artificial Intelligence and Soft Computing*. Springer, 2018, pp. 724–735.

[12] M. Kaden, M. Lange, D. Nebel, M. Riedel, T. Geweniger, and T. Villmann, “Aspects in classification learning-review of recent developments in learning vector quantiza-

tion," *Foundations of Computing and Decision Sciences*, vol. 39, no. 2, pp. 79–105, 2014.

[13] A. Boubezoul, S. Paris, and M. Ouladsine, "Application of the cross entropy method to the glvq algorithm," *Pattern Recognition*, vol. 41, no. 10, pp. 3173–3178, 2008.

[14] N. R. Pal, K. Pal, J. M. Keller, and J. C. Bezdek, "A possibilistic fuzzy c-means clustering algorithm," *IEEE transactions on fuzzy systems*, vol. 13, no. 4, pp. 517–530, 2005.

[15] R. Krishnapuram and J. M. Keller, "A possibilistic approach to clustering," *IEEE transactions on fuzzy systems*, vol. 1, no. 2, pp. 98–110, 1993.

[16] T. Villmann, A. Bohnsack, and M. Kaden, "Can learning vector quantization be an alternative to svm and deep learning?: recent trends and advanced variants of learning vector quantization for classification learning," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 7, 2017.

[17] S. Seo and K. Obermayer, "Soft learning vector quantization," *Neural computation*, vol. 15, no. 7, pp. 1589–1604, 2003.

[18] B. Hanczar, "Performance visualization spaces for classification with rejection option," *Pattern Recognition*, vol. 96, p. 106984, 2019.

[19] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[20] J. Ravichandran, "Prototorch," https://github.com/si-cim/prototorch, 2020.

[21] W. N. Street, W. H. Wolberg, and O. L. Mangasarian, "Nuclear feature extraction for breast tumor diagnosis," in *Biomedical image processing and biomedical visualization*, vol. 1905. International Society for Optics and Photonics, 1993, pp. 861–870.

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, Datum im Vorspann festlegen!