CHAPTER 4

DESIGN OF PREPROCESSING ALGORITHMS

A major challenge faced by telecom industries during customer loyalty assessment for predicting churners is to accurately and efficiently analyze the growing volumes of customer data. Data mining is a powerful tool that enables analysts who may lack extensive training as data analysts to explore large databases quickly and efficiently. The usage of data mining during churn prediction is motivated by the following two facts:

1. Data becomes information when it is effectively analyzed.
2. Information becomes knowledge when it is effectively interpreted.

Thus, to convert raw customer data into information and knowledge, several data mining techniques are proposed in this research work. Out of these techniques, the focus of the first phase is two-fold, handling missing data and also outlier detection and treatment.

Correct and accurate data is a key prerequisite for successful data mining and knowledge discovery. Previous reports analyzing the importance of preprocessing suggests that data preparation takes 60 to 80% of the time involved in a data mining study. Data preprocessing is an expensive and labour intensive task and therefore has to be performed once before analysis. A data preprocessing is a process that consists of data cleaning, data integration and data transformation which is usually processed by a computer program. It intends to reduce some noises, incomplete and inconsistent data. The results from preprocessing step are later used by data mining algorithms.

Two important tasks of data preprocessing are missing value handling and outlier detection. In missing value handling procedures, missing data values are predicted or imputed to form a complete dataset that can be used by data mining applications. The second task, outlier detection and treatment, is
another very important part of any modeling system. The basic definition of an outlying observation is a data point or points that do not fit the model of the rest of the data.

4.1. MISSING VALUE HANDLING

The missing value handling algorithm of the research work focuses only on those attributes that are related to customer data helping to gain knowledge on churners affecting the final output of the mining processes. The cleaning of data is performed in two steps.

Step 1: Implement a Recursive routine that removes data records that are not important for analysis
Step 2: Implement a missing value handling procedure that fills in missing data items or records in the prime dataset.

The telecom databases considered in the study consist of personal, plan details and billing details of a customer. Personal details include name, address, phone and employment details whereas the plan details include information regarding the type of subscription along with the services provided, and billing details consists of the information regarding call and payment details.

First the proposed missing value routine performs a global analysis and removes all those records for which all attributes regarding customer personal information, call and payments details are empty. These customers are not potential customers and therefore are safely removed. The second step of the proposed missing value algorithm performs attribute level analysis for missingness treatment. Here, each attribute is treated separately and empty values are estimated. Careful scrutiny of the database revealed that missing values in the state field in customer address and billing details has maximum impact during churn prediction. The first task of Phase I proposes a simple recursive algorithm for handling state field empty values, while an enhanced K-Nearest Neighbour (KNN) algorithm is proposed to solve the problem of
missingness in the billing attributes. Details regarding these algorithms are given in this section.

4.1.1. State Attribute

All telecommunication companies have a master database consisting of area, state and its corresponding codes. This master table is used as a code book by the proposed algorithm. The algorithm, when encountering a NULL (empty value) in the area code or state code field, simply performs a recursive search in the master table and selects the corresponding code while using the area / state name as search key. If no match is found, then the entire record is deleted. If the area / state name itself is missing, then a search in the other address fields like street name is performed. If not available, then the entire record is deleted, else a similar search is performed with street name and matching area / state code are retrieved and used to fill the empty fields.

4.1.2. Billing Details Attributes

Missing values in billing details have a serious impact on the performance of loyal customers and churners prediction. The algorithm enhanced the conventional KNN imputation algorithm by combining it with a generalized learning vector quantization method.

A) Conventional KNN Method

In this method, the missing values of an instance are imputed by considering a given number of instances that are most similar to the instance of interest. The similarity of two instances is determined using a distance function. The algorithm is as follows

1. Divide the data set D into two parts. Let $D_m$ be the set containing the instances in which at least one of the features is missing. The remaining instances with complete feature information form a set called $D_c$.

2. For each vector $x$ in $D_m$:
a) Divide the instance vector into observed and missing parts as \( x = [x_o, x_m] \).

b) Calculate the distance between the \( x_o \) and all the instance vectors from the set \( D_c \). Use only those features in the instance vectors from the complete set \( D_c \), which are observed in the vector \( x \).

c) Use the K closest instances vectors (K-nearest neighbors) and perform a majority voting estimate of the missing values for categorical attributes. For continuous attributes replace the missing value using the mean value of the attribute in the k-nearest neighborhood. The median could be used instead of the mean.

The advantages of KNN imputation are:

1. k-nearest neighbor can predict both qualitative attributes (the most frequent value among the k nearest neighbors) and quantitative attributes (the mean among the k nearest neighbors).
2. It need not create a predictive model for each attribute with missing data. Actually, the k-nearest neighbor algorithm does not create explicit models.
3. It can easily treat instances with multiple missing values.
4. It takes into consideration the correlation structure of the data.

The conventional KNN imputation method (Dixon, 1979) faces some challenges as listed below.

1. Distance metric used
2. High search time
3. The choice of k, the number of neighbors.
4. Performance degradation with huge sized datasets
The first problem is solved by using a feature-weighted distance metric based on Mutual Information (MI), which is a good indicator of dependence between the random variables. The second problem is solved by using Learning Vector Quantization (LVQ) combined with generalized relevance learning to reduce the search time during missing value treatment. The third problem is concerned with the choice of k (number of neighbors). Experiments showed that a value of 10 produce best results in terms of accuracy and hence is used in further experimentation. The fourth problem is solved by using a partition algorithm that divides the original dataset into small subsets. The enhanced KNN imputation method is applied on each small datasets and the results are merged to form the complete dataset.

**B) Learning Vector Quantization (LVQ)**

Distance-based classification schemes can be implemented efficiently in the framework of the popular learning vector quantization (LVQ). LVQ systems are flexible, easy to implement, and can be applied straightforward in multi-class problems. Because LVQ prototypes are determined in the feature space of observed data, the resulting classifiers can be interpreted intuitively. LVQ classifiers are widely used in a variety of areas including image processing, medical applications, control of technical processes, and bioinformatics.

LVQ has been widely used to improve the performance of classifiers in data mining applications (Mattfeldt et al., 2004; Peres and Pedreira, 2009; Chen et al., 2011). In the present research work, this is combined with simultaneous missing value imputation and the process is explained in the following sections.

Self-organizing methods such as the self-organizing map (SOM) or vector quantization (VQ) as introduced by Kohonen provide a successful and intuitive method of processing data for easy access. Assumed data are labeled, an automatic clustering can be learned via attaching maps to the SOM or

A common feature of unsupervised algorithms and LVQ is the fact that information is provided by the distance structure between the data points which is determined by the chosen metric. Learning heavily relies on the commonly used Euclidean metric and depends on the fact that the Euclidean metric is appropriate for the respective learning task. Therefore data are to be preprocessed and scaled such that the input dimensions have the same importance for the classification. In particular, the important features for the respective problem are to be found, which is usually done by experts or with rules of the thumb. This process is time consuming and requires prior knowledge which is often not available. Hence methods have been proposed which adapt the metric during training. As an example Distinction sensitive LVQ (DSLVQ), automatically determines weighting factors to the input dimensions of the training data (Pregenzer et al., 1996). The algorithm adapts LVQ3 for the weighting factors according to plausible heuristics.

LVQ combines the elegance of simple and intuitive updates in unsupervised algorithms with the accuracy of supervised methods. In this present work it is used as a pruning algorithm to identify irrelevant data. The main idea here is to introduce weighting factors to the data dimensions which are adapted automatically such that the classification error becomes minimal. In particular, the relevance learning capability of LVQ is exploited.

C) Generalized LVQ (GLVQ)

Standard LVQ as proposed by Kohonen (1997) is a heuristic approach to reduce the classification error in supervised learning. However, the adaptation dynamic does not minimize any continuous cost function and shows instabilities. The result of LVQ crucially depends on the initialization of the
prototypes, commonly initialized with the center points of the classes or random representatives from the training set. The algorithms use iterative local learning rules which can easily get stuck in local optima. GLVQ avoids the numerical instabilities of LVQ due to a stochastic gradient descent on a cost function which optimizes the margin (Crammer et al., 2002). However, it depends on the initialization process.

To overcome this drawback a combination of GLVQ with Neural Gas (NG) has been proposed. Generalized Learning Vector Quantization (GLVQ), introduced by Sato and Yamada (1995), is an extension of the basic LVQ-algorithms that combines relevance learning and neighborhood cooperativeness (Supervised Relevance Neural Gas - SRNG). Supervised Neural Gas is considered as a representative for prototype based classification approaches. It can be combined with the demanded feature of relevance learning. Moreover, it is a stochastic gradient descent algorithm, which is a margin optimizer with known bounds of generalization ability (Hammer et al., 2005).

It consists of a cost function to minimize the trough of the learning rule (Villmann and Hammer, 2002). This cost function leads to training similar to NG or simple GLVQ, respectively; depending on the choice of a parameter of the cost function. During training, this parameter is varied so that neighborhood cooperation assures a distribution of the prototypes among the data set at the beginning and a good separation of the classes can be obtained at the end of training.

The formal description was provided by Villmann et al., (2006) and is reproduced in this section. Let \( c_v \in L \) be the label of input \( v \), \( L \) a set of labels (classes) with \( \#L = N_L \). Let \( V \subseteq R^{DV} \) be a finite set of inputs \( v \). LVQ uses a fixed number of prototypes (weight vectors, codebook vectors) for each class. Let \( W = \{ w_r \} \) be the set of all codebook vectors and \( c_r \) be the class label of \( w_r \). Furthermore, let \( W_c = \{ w_i | c_r = c \} \) be the subset of prototypes assigned to class \( c \in L \).
The task of vector quantization is realized by the map $\Psi$ as a winner-take-all rule, i.e. a stimulus vector $v \in V$ is mapped onto that neuron $s \in A$ the pointer $w_s$ of which is closest to the presented stimulus vector $v$,

$$
\Psi_v^\lambda : v \rightarrow s(v) = \arg \min_{r \in A} d^\lambda(v, w_r)
$$

(4.1)

with $d^\lambda(v, w)$ being an arbitrary differentiable similarity measure which may depend on a parameter vector $\lambda$, where $\lambda$ is assumed to be fixed. The neuron $s(v)$ is called winner or best matching unit. The subset of the input space

$$
\Omega_r^\lambda = \{ v \in V : r = \Psi_{V \rightarrow A}(V) \}
$$

(4.2)

which is mapped to a particular neuron $r$ according to Equation (4.1), forms the (masked) receptive field of that neuron forming a Voronoi tessellation. If the class information of the weight vector is used, the boundaries $\partial \Omega_r^\lambda$ generate the decision boundaries for classes. A training algorithm should adapt the prototypes such that for each class $c \in L$, the corresponding codebook vectors $W_c$ represent the class as accurately as possible. This means that the set of points in any given class $V_c = \{ v \in V | c_v = c \}$, and the union $U_c = \bigcup_{r | w_r \in W_c} \Omega_r$ of receptive fields of the corresponding prototypes should differ as little as possible.

The main idea behind GLVQ is to introduce a cost function such that the learning rule gives a gradient descent on it. At the same time it should assess the number of misclassifications of the prototype based classification. Let $f(x) = (1 + \exp(-x))^{-1}$ be the logistic function. GLVQ minimizes the cost function

$$
\text{Cost}_{\text{GLVQ}} = \sum_v f(\mu^\lambda_v(v))
$$

(4.3)

$$
\mu^\lambda_v(v) = \frac{d^\lambda_{r+} - d^\lambda_{r-}}{d^\lambda_{r+} + d^\lambda_{r-}}
$$

(4.4)
through stochastic gradient descent, where \( d_{r_+}^\lambda \) is the squared distance of the input vector \( v \) to the nearest codebook vector labeled with \( c_{r_+} = c_v \) (\( w_{r_+} \)) and \( d_{r_-}^\lambda \) is the squared distance to the best matching prototype but labeled with \( c_{r_-} \neq c_v \), (\( w_{r_-} \)). As it was shown by Sato and Yamada (1998), the usage of the function \( \mu_\lambda(v) \) yields robust behavior of LVQ.

The learning rule of GLVQ is obtained taking the derivatives of the above cost function. Using Equation (4.5) with Equation (4.6) the weight updates are obtained (Equations 4.7 and 4.8).

\[
\frac{\partial \mu_\lambda(v)}{\partial w_{r_+}} = \xi^+ \frac{\partial d_{r_+}^\lambda}{\partial w_{r_+}} \quad \text{and} \quad \frac{\partial \mu_\lambda(v)}{\partial w_{r_-}} = \xi^- \frac{\partial d_{r_-}^\lambda}{\partial w_{r_-}}
\]  

\[
\xi^+ = \frac{2d_{r_+}^\lambda}{(d_{r_+}^\lambda + d_{r_-}^\lambda)^2} \quad \text{and} \quad \xi^- = \frac{2d_{r_-}^\lambda}{(d_{r_+}^\lambda + d_{r_-}^\lambda)^2}
\]  

\[
\Delta w_{r_+} = \epsilon^+ f|_{\mu_\lambda(v)} \xi^+ \frac{\partial d_{r_+}^\lambda}{\partial w_{r_+}}
\]  

\[
\Delta w_{r_-} = -\epsilon^- f|_{\mu_\lambda(v)} \xi^- \frac{\partial d_{r_-}^\lambda}{\partial w_{r_-}}
\]

where \( \epsilon^+, \epsilon^- \) are learning rates. The original (unsupervised) Neural Gas (NG) adapts unlabeled prototypes \( w_r \in W \) according to a given data set such that the cost function

\[
\text{Cost}_{NG}(\gamma) = \frac{1}{C(\gamma, \mathbb{K})} \sum_r r_1(\gamma)
\]  

is minimized with local costs

\[
\epsilon_r(\gamma) = \int P(v) \cdot h_r(r, v, W) (v - w_r)^2 \, dv
\]

and neighborhood function known from NG:
Thereby $k_r(v, W)$ yields the number of prototypes $w_r$ for which the relation $d^h(v, w_r) \leq d^h(v, w_i)$ is valid, i.e. $k_r(v, W)$ is the winner rank (Martinetz et al., 1993). $C(\gamma, K)$ is a normalization constant depending on the neighborhood range $\gamma$ and the cardinality $K$ of $W$. The learning rule reads as

$$\Delta w_r = \epsilon \cdot h_\gamma(r, v, W) (v - w_i)$$

(4.12)

minimizing the cost function. The initialization of the prototypes is not longer crucial in NG because of the involved neighborhood cooperation.

As mentioned previously, Supervised Neural Gas (SNG) constitutes a combination of GLVQ and NG. Again, let $W_c = \{w_i | c_r = c\}$ be the subset of prototypes assigned to class $c \in L$ and $K_c$ its cardinality. Further let there be $m$ data vectors $v_i$. The neighborhood learning for a given input $v_i$ with label $c$ is applied to the subset $W_c$. The respective cost function is

$$\text{Cost}_{\text{SNG}}(\gamma) = \sum_{i=1}^{m} \sum_{r|w_r \in W_{c_i}} \frac{h_\gamma(r, v_i, W_{c_i}) f(\mu_\lambda (r, v))}{C(\gamma, K_{c_i})}$$

(4.13)

with $f(x) = (1 + \exp (-x))^{-1}$ and according to equation (4.4), $d^h_{r-}$ is defined as in GLVQ above and $d^h_r = d^h(v, w_i)$.

The neighborhood cooperativeness makes sure that prototypes are spread among data of their respective classes with the $\lim_{\gamma \to 0} \text{Cost}_{\text{SNG}}(\gamma) = \text{Cost}_{\text{GLVQ}}(\gamma)$ holds. Hence, for vanishing neighborhood the SNG also becomes optimal in the sense of margin analysis. However, if the neighborhood range $\gamma$ is large, typically at the beginning of the training, the prototypes of one class share their responsibilities no a given input. The neighborhood cooperation is involved such that initialization of the prototypes is no longer crucial. Given a training example $(v_i, c_i)$ all prototypes $w_r \in W_{c_i}$ and the closest wrong prototype
are adapted. Considering Equation (4.14), the update rules are obtained using Equations (4.15) and (4.16). The neighborhood co-operativeness is applied only to the correct prototypes.

\[
\xi^+_r = \frac{2d^\lambda_{r-}}{(d^\lambda_r + d^\lambda_{r-})^2} \quad \text{and} \quad \xi^-_r = \frac{2d^\lambda_r}{(d^\lambda_r + d^\lambda_{r-})^2}
\]  

\[
\Delta w_r = \epsilon^+ \xi^+_r \frac{f^\gamma_{\mu\lambda}(r,v) h_{\gamma}(r,v_i,w_{c_i})}{C(\gamma,K_{c_i})} \frac{\partial d^\lambda_r}{\partial w_r}
\]  

\[
\Delta w_r = -\epsilon^- \sum \xi^-_r \frac{f^\gamma_{\mu\lambda}(r,v) h_{\gamma}(r,v_i,w_{c_i})}{C(\gamma,K_{c_i})} \frac{\partial d^\lambda_r}{\partial w_r}
\]  

Margin analysis of an algorithm is important to access the level of confidence of a classifier with respect to its decision. For example, the sample margin is defined as the distance between the input and the decision boundary. The natural choice of this margin to be maximized in learning vector quantization causes numerical instabilities. An alternative definition gives the hypothesis margin: this margin is the distance that the classifier can move without changing the way it labels any sample data. The hypothesis margin of a prototype based classifier is given by Equation (4.17).

\[
(d^\lambda_{r+})^{1/2} - (d^\lambda_{r-})^{1/2}
\]

In fact, the GLVQ (and, hence, SNG for vanishing neighborhood) maximizes a cost function closely related to this hypothesis margin and, hence, can be taken as maximum margin algorithm provided the similarity measure can be interpreted as a kernelized version of the Euclidean metric, i.e. the similarity measure is symmetric and its negative is conditionally positive definite. Further, any given margin provides an upper bound for the generalization error of the classifier such that the higher the margin the lower the generalization error.
This paragraph describes the relevance learning procedure, referred to as SRNG. The relevance learning algorithm studies the influence of the parameter vector \( \lambda = (\lambda_1, ..., \lambda_m) \) in the distance measure \( d^\lambda(v, w) \). Focus is in adaptation of the distance measure in dependence on the \( \lambda_k \) to minimize the cost function. After finding the relevance, an adaptation step for the parameters \( \lambda_k \) is added to the usual weight vector adaptation. Thus, \( \lambda_k \geq 0 \) and \( \sum_k \lambda_k = 1 \), where \( \lambda_k \) is given as

\[
\Delta \lambda_k = -2 \varepsilon \lambda \sum_{r|w_r \in W_c} \frac{f^r_{|\mu_\lambda(r,v)h_\gamma(r,v_i,w,c_i)}}{C(\gamma,kc_i)} \left( \xi_r^+ \frac{\partial d_r^\lambda}{\partial \lambda_k} - \xi_r^- \frac{\partial d_r^\lambda}{\partial \lambda_k} \right) \quad (4.18)
\]

followed by a renormalization. \( \varepsilon \lambda > 0 \) is the learning rate. For vanishing neighborhood cooperativeness \( \gamma \to 0 \) the SRNG turns to Generalized Relevance LVQ (GRLVQ) as relevance learning in GLVQ with

\[
\Delta \lambda_k = -\varepsilon \lambda f^r_{|\mu_\lambda(r,v)} \frac{\partial \mu_\lambda(r,v)}{\partial \lambda_k} \quad (4.20)
\]

The learning rate \( \varepsilon \lambda \) should be in both approaches at least a magnitude smaller than the learning rates for the weight vectors. Then the weight vector adaptation takes place in a quasi stationary environment with respect to the (slowly) changing metric. Hence, the margin optimization takes place for each level of the parameter \( \lambda_k \) and overall optimization of margin including a relevance weighting of the parameters is obtained.

As pointed out before, the similarity measure \( d_\lambda(v, w) \) is only required to be differentiable with respect to \( \lambda \) and \( w \). The triangle inequality has not to be fulfilled necessarily. This leads to a great freedom in the choice of suitable measures and allows the usage of non-standard metrics in a natural way. In particular, kernel based similarity measures are allowed.
In this way SNG/SRNG are comparable to SVMs. The margin analysis of this LVQ-version also holds for adaptive relevance metrics and kernelizations.

\[ d^\lambda(v, w) = \sum_{i=1}^{D_y} \lambda_i (v_i - w_i)^2 \]  

(4.21)

is the squared, scaled Euclidean distance, whereby again \( \lambda_i \geq 0 \) and \( \sum \lambda_i = 1 \), (5.7) and (5.8) is immediately obtained for GLVQ update:

\[ \Delta w_r^+ = e^+ \cdot 2 f'_{|\mu_\lambda(v)} \xi^+ \Lambda (v-w^+_r) \]  

(4.22)

\[ \Delta w_r^- = e^- \cdot 2 f'_{|\mu_\lambda(v)} \xi^- \Lambda (v-w^-_r) \]  

(4.23)

respectively, with \( \Lambda \) being the diagonal matrix with entries \( \lambda_1, \ldots, \lambda_{DV} \) and the relevance parameter update Equation (4.20) in GRLVQ is given as

\[ \Delta \lambda_k = -e_\lambda f'_{|\mu_\lambda(v)} (\xi^+ (v-w^+_r)_k^2 - \xi^- (v-w^-_r)_k^2) \]  

(4.24)

with \( k = 1 \ldots DV \). In case of SNG and SRNG, the updates in case of the scaled Euclidean distance are obtained as

\[ \Delta w_r = 2e^+ \xi^+_r \frac{f'_{|\mu_\lambda(r,v)} h_\gamma(r,v_i,w_{c_i})}{C(\gamma, Kc_i)} \Lambda(v-w_r) \]  

(4.25)

\[ \Delta w_r = 2e^- \sum_{r|w_r \in Wc} \xi^-_r \frac{f'_{|\mu_\lambda(r,v)} h_\gamma(r,v_i,w_{c_i})}{C(\gamma, Kc_i)} \Lambda(v-w^-_r) \]  

(4.26)

with \( \Lambda \) being again the diagonal matrix with entries \( \lambda_1, \ldots, \lambda_{DV} \) and

\[ \Delta \lambda_k = e_\lambda \sum_{r|w_r \in Wc} \frac{f'_{|\mu_\lambda(r,v)} h_\gamma(r,v_i,w_{c_i})}{C(\gamma, Kc_i)} ((\xi^+_r (v-w_r)_k^2 - \xi^-_r (v-w^-_r)_k^2) \]  

(4.27)
D) **Proposed Method**

The proposed enhanced KNN algorithm initially partitions the dataset into two groups, one having instances with Missing Values (MV) and another having Complete Value (CV). In Step 2, all records in MV that have no relevancy are removed. The proposed enhanced KNN Imputation algorithm is applied to the resultant dataset, MV’, to estimate the missing values. Let this be IMV. The new group with imputed value (IMV) is then combined with the instance group (CV) to obtain a complete dataset. The proposed algorithm is presented in Figure 4.1.

- Let $D_m$ be MV’ and $D_c$ be CV
- Estimate Mutual Information (MI) estimation between each input attribute and the target class variable
- Use Generalized Learning Vector Quantization during learning process that uses MI to identify relevant and irrelevant input feature. A high MI between an input feature and the target means that this feature is relevant. Otherwise, the input feature is irrelevant.
- Perform Enhanced KNN
  - Divide $D_m$ into observed ($x_o$) and missing ($x_m$) parts
  - Calculate the distance between the $x_o$ and all the instance vectors from the set $D_c$ using Heterogeneous Euclidean-Overlap Metric (HEOM)
  - Assign weight to each nearest neighbour (nearest neighbour weight = 1 and furthest neighbour weight is scaled to a value between 0 and 1).
- Use K-nearest neighbors and perform a majority voting estimate of the missing values for categorical attributes. For continuous attributes replace the missing value using the mean value of the attribute in the k-nearest neighborhood.

*Figure 4.1 : Enhanced KNN Imputation Algorithm*
(i) **Distance Metric**

To classify an unlabeled pattern (missing value) \( x \), the distances from \( x \) to the labeled instances are computed, its \( K \) nearest neighbors are identified, and the class labels of these nearest neighbors are then used to determine the class label of \( x \). According to the voting KNN rule, \( x \) is assigned to the class represented by a majority of its \( K \) nearest neighbours. In the standard KNN algorithm, the \( K \) neighbours are implicitly assumed to have equal weight in decision, regardless of their distances to the pattern to be classified. Some approaches have been proposed based on assigning different weights to the \( K \) neighbours according to their distances to \( x \), with closer instances having greater weights.

The enhanced KNN imputation method uses Heterogeneous Euclidean-overlap metric (HEOM) to calculate the distance between two vectors. The reason for using HEOM over the traditional distance metrics is that the dataset might contain the datasets in the real world often contain varied or mixed attributes which needs to be handled separately to achieve better efficiency as explained below.

The approach for missing data imputation is a modified KNN imputation method. The KNN imputation method is modified because, the conventional method’s learning process is not oriented to provide an appropriate imputed dataset for solving classification task. The modified method uses an effective procedure where the neighbourhood is selected by considering the input attribute relevance for classification. For each incomplete pattern, its selected \( K \) neighbours are used to provide imputed values which can make the classifier design easier, and thus, the classification accuracy is increased. This approach uses a feature-weighted distance metric based on MI, which is a good indicator of dependence between random variables.
The distance between two input vectors $x_a$ and $x_b$ is denoted as $d(x_a, x_b)$ and is calculated using Equation (4.28).

$$d(x_a, x_b) = \sqrt{\sum_{j=1}^{n} d_j(x_{aj}, x_{bj})^2}$$  \hspace{1cm} (4.28)

where $d_j(x_{aj}, x_{bj})$ is the distance between $x_a$, $x_b$ on its $j$th attribute and is given by Equation (4.29).

$$d_j(x_{aj}, x_{bj}) = \begin{cases} 
1 & (1 - m_{aj})(1 - m_{bj}) = 0 \\
0 & x_j \text{ is qualitative} \\
(d_N(x_{aj}, x_{bj}) & x_j \text{ is quantitative} 
\end{cases}$$  \hspace{1cm} (4.29)

When the input values are unknown, then it is treated missing data and a distance value of 1 is returned in these situations. The overlap function $d_0$ is assigned a value 0 if the qualitative features are same, otherwise it is assigned a value 1 (Equation 4.30) and $d_N$ is the range normalized difference distance function and is given by Equation (4.31).

$$d_0(x_{aj}, x_{bj}) = \begin{cases} 
0 & \text{if } x_{aj} = x_{bj} \\
1 & \text{if } x_{aj} \neq x_{bj} 
\end{cases}$$  \hspace{1cm} (4.30)

$$d_N(x_{aj}, x_{bj}) = \frac{|x_{aj} - x_{bj}|}{\max(x_j) - \min(x_j)}$$  \hspace{1cm} (4.31)

where $\max(x_i)$ and $\min(x_i)$ are the maximum and minimum values observed in the N training instances for the $j$th attribute.

Following the distance-weighted rule proposed by Dudani (1976) and weighting procedure described in Brown (2002), a weight $a_k$ is assigned to each nearest neighbour $v_k$ of $x$, with $k = 1, 2, \ldots, K$. The nearest neighbour receives a weight of 1, the furthest neighbour a weight of 0, and the remaining neighbours are scaled linearly between 0 and 1. An unlabeled pattern is
assigned to the class producing the highest summed weight among its reference neighbours.

(ii) **Mutual Information (MInf)**

Let \( Y \) and \( Z \) be two discrete random variables. The entropy is a measure of uncertainty of random variables. Making use of the Shannon’s information theory, if \( Y \) has alphabet \( Y' \) and the probability density function (pdf) is \( p(y) = \Pr\{Y = y\}, y \in Y' \) the entropy of \( Y \) is defined as in Equation (4.32).

\[
H(Y) = - \sum_{y \in Y'} p(y) \log p(y) \tag{4.32}
\]

The conditional entropy measures the resulting uncertainty on \( Z \) knowing \( Y \), and is given by

\[
H(Z \mid Y) = - \sum_{y \in Y', z \in Z'} p(y, z) \log p(z \mid y) \tag{4.33}
\]

where \( p(y) \) and \( p(z) \) are two pdfs of two different random variables and \( p(y, z) \) is the joint pdf of \( Y \) and \( Z \) and \( p(z \mid y) \) is the conditional pdf of \( Z \) given \( Y \). Formally, the MInf of \( Y \) and \( Z \) is defined as:

\[
I(Y; Z) = \sum_{y \in Y', z \in Z'} p(y, z) \log \frac{p(y, z)}{p(y)p(z)} \tag{4.34}
\]

For continuous random variables, the summation is replaced by a definite double integral

\[
I(Y; Z) = \int \int_{YZ} p(y, z) \log \frac{p(y, z)}{p(y)p(z)} \, dy \, dz \tag{4.35}
\]

The MI and the entropy have the following relation

\[
I(Y; Z) = H(Z) = H(Z \mid Y) \tag{4.36}
\]
which is the reduction of the uncertainty of \( Z \) when \( Y \) is known. Using the properties of the entropy, the MInf can be rewritten into \( I(Y; Z) = H(Y) + H(Z) - H(Y, Z) \) where \( H(Y, Z) \) is the joint entropy of \( Y \) and \( Z \). The MInf is a natural measure of the dependence between random variables. If the variables \( Y \) and \( Z \) are independent, then \( H(Y, Z) = H(Y) + H(Z) \) and \( H(Z|Y)=H(Z) \), i.e., the MInf of two independent variables is zero. In addition to this, the MInf measures any relationship between variables, contrarily to the Pearson correlation that only measures the linear relations.

- **Computation of the MI in Classification Tasks**

  This work is focused on the MInf estimation between each input attribute and the target class variable. In classification problems, the target has discrete values (\( N_c \) possible classes) while the input features can be qualitative or quantitative variables. According to Francois et al. (2007), the MInf between the \( j^{th} \) input attribute \((x_j)\) and the target class variable \((C)\) is given as

  \[
  I(X_j; C) = H(C) - H(C|X_j)
  \]  

  (4.37)

  In this equation, the entropy measure \( H(C) \) is computed using Equation (4.38)

  \[
  H(C) = - \sum_{c=1}^{N_c} p(c) \log p(c)
  \]  

  (4.38)

  For computing (4.18), \( p(c) \) is estimated by \( p'(c) = N_c/M \), where \( N_c \) is the number of instances belonging to the class \( c \), and \( N \) is the number of training instances. The difference between \( N_c \) and \( N_C \) (number of classes) is noted. The estimation of the conditional entropy \( H(C|X_j) \) depends on the nature of the input attribute \( X_j \). If the \( j^{th} \) input feature is a qualitative variable, \( H(C|X_j) \) can be easily computed using (4.18). In this case, the estimation of the densities for qualitative variables is straightforward by means of the histogram approximation from the training input data. When the input feature is a continuous variable, the conditional entropy is given by Equation (4.19) and is difficult to obtain because the estimation of \( p(c|x_j) \) is not direct.
\[
H(C \mid X_j) = - \int p(x_j) \sum_{c=1}^{N_c} p(c \mid x_j) \log p(c \mid x_j) \, dx_j 
\]  

(4.39)

To solve this, the Parzen window density estimation method is used (Kwak and Choi, 2002). This approach provides an estimation of \( p(x_j) \) as in Equation (4.40).

\[
\hat{p}(x_j) = \frac{1}{N} \sum_{i=1}^{N} \phi(x_j - x_{ij}, h)
\]

(4.40)

where \( \phi(.) \) is the window function and \( h \) is the smoothing parameter. It has been shown that \( \hat{p}(x_j) \) converges to the true density if \( \phi(.) \) and \( h \) are selected properly.

The rectangular and the Gaussian functions are commonly used as window functions (Duda et al., 2000). Once \( \hat{p}(x_j) \) has been computed by the Parzen window approach, the estimate of the conditional pdf is written according to the Bayesrule (Equation 4.41).

\[
\hat{p}(c \mid x_j) = \frac{\hat{p}(x_j \mid c)p(c)}{\hat{p}(x_j)}
\]

(4.41)

For each class, \( \hat{p}(x_j \mid c) \) is obtained using the Parzen window approach:

\[
\hat{p}(x_j \mid c) = \frac{1}{N_c} \sum_{i \in I_c} \phi(x_j - x_{ij}, h)
\]

(4.42)

Where \( I_c \) is the set of indexes of the training patterns labeled with class \( c \). Finally, if the integration in (4.20) is replaced with a summation of the training instances, the estimate of \( H(C \mid x_j) \) can be achieved using Equation (4.43).

\[
H(C \mid X_j) = - \sum_{i=1}^{N} \hat{p}(x_j) \sum_{c=1}^{N_c} \hat{p}(c \mid x_j) \log \hat{p}(c \mid x_j)
\]

(4.43)
Thus using these equations and the estimated densities Minf can be estimated.

- **Feature relevance rating based on MI**

  Feature selection algorithms, in general, assign binary weights to features, a weight equal to 1 for selected relevant attributes; a value of 0 for irrelevant features. This method has the advantage of reducing input dimensionality, computation complexity and time. In GL-model, the feature-weighted procedure assigns one weight per feature according to the MInf estimate between each feature and the target class variable. For classification, the MInf measures the amount of information contained in an input feature for predicting the target class variable. A high MI between an input feature and the target means that this feature is relevant, regardless of the classification algorithm. Otherwise, when the shared information between both variables is small, the input feature is irrelevant for the classification task.

  The MI concept to weight the input features distances according to their relevance for classification assigns a weight \( \lambda_j \) to each jth input feature according to the amount of information that this attribute contains about the target class variable. The scaling factors \( \lambda_j \) is computed heuristically, as given by Weinberger *et al.* (2006). The higher \( \lambda_j \) the more relevant \( X_j \) is for classification.

  \[
  \lambda_j = \frac{I(X_j, C)}{\sum_{f=1}^{n} I(X_j, C)} \tag{4.44}
  \]

  According to the MI concept, the feature-weighted distance metric (Equation) between two input vectors \( x_a \) and \( x_b \) is computed using Equation (4.45).
\[ d_j(x_a, x_b) = \sqrt{\sum_{j=1}^{n} \lambda_j d_j(x_{a_j}, x_{b_j})^2} \]  

(4.45)

where \( d_j \) is the distance. When this MI-weighted distance metric is implemented in KNNImpute, the modified MI-KNNImpute is formed.

The advantage of this method is that it selects the K-nearest cases by considering the input attribute relevance to the target class. Thus, the method adds useful information about classification task during imputation stage and provides missing data estimation which enhances the classification task.

The MI between an attribute and target class variable is estimated by the Parzen window method. The MI-KNN classify assigns a weight \( \beta_k \) to the k-th nearest neighbour using Equation (4.46).

\[
\beta_k(x) = \frac{d_1(v_k, x) - d_1(v_k, x)}{d_1(v_k, x) - d_1(v_1, x)} = 1 \text{ when } d_1(v_k, x) = d_1(v_k, x) \]  

(4.46)

where \( d_1(.) \) is the distance measure based on MI concept (previous section) and \( V_x \) is the set of K nearest neighbours of \( x \) arranged in increasing order of distance. During classification, \( x \) is assigned to the class for which the weights \( \beta_k \) of the representatives among the \( K_C \) nearest neighbours sum to the largest value. This is performed by considering the relevancy between the input features of the target class variable.

Finally, the number of neighbours (parameter K) used for classification can be different than the K value used for pruning and imputation. KI refers to the K nearest neighbours used for imputation and KC represents the number of nearest neighbours selected for classification. As the main objective is to solve the classification task, both parameters (i.e., KI and KC) are tuned in order to optimize the classification performance. The optimal values of K to impute (KI) and to classify (KC) are selected by 10-fold cross validation, i.e., by means
of the classification accuracy results on the validation set. All the experimental results in this chapter are averaged on 10-fold cross validation repetition.

Upon careful scrutiny, it was found that the feature selection and relevancy checking algorithm of enhanced KNN Imputation method can further be tuned to save time and improve classification accuracy. During experimentation, it was found that efficient selection of $K_1$ and $K_c$ values for feature selection is most important. Incorrect values often resulted in degraded results and much effort was focused on finding these optimal values through multiple execution of the program. In order to solve this problem, the present research work uses LVQ (Learning Vector Quantization) method with generalized relevance learning (Hammer and Villman, 2002; Villman et al., 2006) to improve the enhanced KNN Imputation method. The proposed model is called as EKDDI-LVQ (Enhanced KDD Imputation with LVQ).

(iii) Classification Process

The information theoretic classification is based on maximization of the mutual information between the class information and the data to approximate the Bayes error. The basic idea is the usage of the mutual information between the class labels of input vectors taken as a random variable and the network output. This approach is based on the following considerations: The ultimate criterion for classification tasks is the Bayes error. Yet, mutual information can be taken as a proper approximation: Generally, the mutual information $I(X,Y)$ measures the information transfer between the random variables $X$ and $Y$

\[
I(Y, X) = H(Y) + H(X) - H(Y, X) \\
= H(Y) + H(Y|X) \\
= H(X) + H(X|Y)
\]

(4.47)
with \( H(Y) \) usually being the Shannon entropy and \( H(Y \mid X) \) its conditional counterpart. As pointed out by Torkkola (2003), the Bayes error \( B(Y) \) has an upper bound given by

\[
B(Y) \leq \frac{1}{2} H(X \mid Y) = \frac{1}{2} (H(X) - I(Y, X))
\]

(4.48)

and a lower bound given by Fano’s inequality

\[
P(Y \neq \hat{Y}) \geq \frac{H(Y \mid X) - 1}{\log(N)} = \frac{H(Y) - I(Y, X) - 1}{\log(N)}
\]

(4.49)

with \( N \) is the number of possible instances of \( Y \) and \( \hat{Y} \) is a certain fixed instance of \( Y \). Hence, maximizing \( I(Y,X) \) is equivalent to minimizing both bounds. Therefore, the mutual information can be taken as an estimate for the Bayes error. Now we can formulate the problem of classification in the following way: Let

\[
y_i = g(v_i, w)
\]

(4.50)

be a transfer function for a given input \( v_i \in V \) with weights \( w \). Then the formal stochastic gradient approach according to the mutual information \( I \) can be written for the weights \( w \) as

\[
\Delta w = \varepsilon \frac{\partial I}{\partial w} \quad \text{with} \quad \frac{\partial I}{\partial w} = \sum_i \frac{\partial I}{\partial y_i} \frac{\partial y_i}{\partial w}
\]

(4.51)

Hence, the task is to determine \( \partial I / \partial y_i \). For this purpose, consider \( I(Y,X) \) and specify \( X = L \), where \( L \) is the set of labels (classes) with \( \#L = N_L \). The motive is to maximize the mutual information between the random sequence of labels \( c_v \) according to the input sequence of \( v \) and the variable \( y \) by adaptation of the weights \( w \).
For high-dimensional large data sets the computational costs become intractable. Therefore, a more convenient approximation is necessary. The key idea is to replace the Shannon entropy in the definition of the mutual information for tractable computation. According to Kapur (1994) the third axiom of additivity/ recursiveness of the Shannon entropy is not necessary, if one is only interested to maximize or minimize the entropy of a system. Therefore, other entropy measures can be used in this case. For easier numerical computation Renyi’s entropy is considered (Renyi, 1961).

\[
H_{\text{Renyi}}(\alpha) = \frac{1}{1-\alpha} \ln(\sum_{l \in \mathbb{L}} (p_l)^{\alpha})
\]

(4.52)

instead of the Shannon entropy for use in the mutual information. For \(\alpha = 2\) it is called quadratic entropy. Then, the mutual information can be written as:

\[
I(Y, L) = \sum_{l \in \mathbb{L}} \int p_l^2(l, y) \, dy + \left( \sum_{l \in \mathbb{L}} p_l^2 \right) \int p^2(y) \, dy - 2 \sum_{l \in \mathbb{L}} p(l) \int p(l, y)p(y) \, dy
\]

(4.53)

with \(p_l\) the a priori probabilities of class \(l\). The probability density is estimated using Parzen window approximation of spherical Gaussian kernels \(G\) as

\[
p(y) = \frac{1}{N} \sum_{i=1}^{N} G(y - y_i, \sigma^2 l)
\]

(4.54)

with the Gaussian kernel in \(d\)-dimensional space defined as

\[
G(y, \Sigma) = \frac{1}{(2\pi)^d/2 |\Sigma|^{1/2}} \exp \left( -\frac{1}{2} y^T \Sigma^{-1} y \right)
\]

(4.55)

Thus, the mutual information \(I\) can be estimated as
\( I(Y, L) \approx \frac{1}{N^2} \sum_{l=1}^{N_L} \sum_{i,j=1}^{N_1} G(yl_l - yl_j, 2\sigma^2.1) + \frac{1}{N^2} \left( \sum_{l=1}^{N_L} p_l^2 \right) \sum_{i,j=1}^{N} G(y_i - y_j, 2\sigma^2.1) \)

\[-2 \frac{1}{N^2} \sum_{l=1}^{N_L} p_l \sum_{i=1}^{N_1} \sum_{j=1}^{N} G(yl_l - y_j, 2\sigma^2.1) \]

(4.56)

using the convolution properties of Gaussian kernels. This formula can be used to carry out the derivative \( \partial_I / \partial_y \). However, the computation is still expensive. Therefore, Torkkola and Campbell, (2000) suggested a stochastic approximation based on the above Parzen estimate. It is assumed for the moment that the database \( V \) only contains two inputs \( v_1 \) and \( v_2 \) with two class possibilities. Then two cases have to be handled separately: both, the transformed data \( y_1 \) and \( y_2 \) come from the same class or from different classes. In the first case it follows that \( I(Y, L) = 0 \) holds. In the second case, \( I(Y, L) \) is given as

\[ I(Y, L) = \frac{1}{4} (G(0, 2\sigma^2.1) - G(y_1 - y_2, 2\sigma^2.1)) \]

(4.57)

which leads to the derivatives

\[ \frac{\partial I}{\partial y_1} = -\frac{1}{8\sigma^2} G(y_1 - y_2, 2\sigma^2.1)(y_2 - y_1) = -\frac{\partial I}{\partial y_2} \]

(4.58)

And the general update rule for this case is given as

\[ \Delta w = \varepsilon \left( \frac{1}{8\sigma^2} G(y_1 - y_2, 2\sigma^2.1)(y_2 - y_1) \left( \frac{\partial y_2}{\partial w} - \frac{\partial y_1}{\partial w} \right) \right) \]

(4.59)

Thus, the information energy is defined as
\[ O(X, Y) = E[Y|X] - E[Y] \]  

(4.60)

where \( E \) is the expectation value. The information energy has the following properties

- \( O(X, Y) \neq O(Y,X) \)
- \( O(X, Y) \geq 0, O(X,Y) = 0 \) iff \( X \) and \( Y \) are statistically independent
- \( O(X, Y) \leq 1 - E[X] \) and \( O(X,Y) = 1 - E[X] \) iff \( X \) is completely dependent on \( Y \)
- \( (X,Y) \) measures the unilateral dependence of \( X \) relative to \( Y \).

Thus the steps in the EKDD-LQV method can be summarized as follows.

Step 1: Read the incomplete dataset
Step 2: Train SRNG with the scaled Euclidean metric (SEM)
Step 3: Initialize relevance factor and \( \lambda_1 \) uniformly and Select relevant neighbours
Step 4: For each incomplete pattern, from the selected \( k \) neighbours impute values

The advantage of this step is that the process of finding mutual information data for identifying relevant values is simplified by using LVQ based SRNG. This when combined with enhanced KNN imputation method increases the accuracy.

4.2. OUTLIER DETECTION

Outliers can arise from several different mechanisms or causes. The two most common causes are,
i) those arising from errors in the data and

ii) those arising from the inherent variability of the data.

It should also be noted that not all outliers are illegitimate contaminants and not all illegitimate scores show up as outliers. It is therefore important to consider the range of causes that may be responsible for outliers in a given data set.

In the present research work, both the scenarios are considered and the concept of outlier detection is used in two manners.

- All erroneous data are assumed to be deviations which do not contribute heavily to the accuracy of the proposed framework and hence are removed during the preprocessing stage.

- Inherent variations that occur randomly and sparsely are considered as abnormal or novel crime activities and are reported as deviations during analysis.

Outliers can be novel outliers (Markou and Singh, 2003a, 2003b), chance discovery outliers (Ohsawa and McBurney, 2003) and noise outliers (Han and Kamber, 2000). Novelty detection outliers are unobserved outliers detected from historical data and whose occurrence reveals an emergent or novel pattern in the data. Example, it might be a new sales trend or a new topic of discussion in a news group. The distinction between novel patterns and outliers is that the novel patterns are typically incorporated into the normal model after being detected.

Chance outliers are outliers which occur by chance and are often indicative either of measurement error or that the population has a heavy-tailed distribution. When it is a measurement error, it has to be handled before the data is incorporated into the model else they indicate that the analyst has to be cautious in handling such data and will be incorporated along with the model.
Noise outliers are data that are not of interest to the analyst and act as hindrance to the data analysis task. Noise outliers can either be removed (Teng et al. 1990) or accommodated (Rousseeuw and Leroy, 1996). In either case, the process has to be performed before data analysis.

Irrespective of the type and cause of outliers, they have to be handled carefully as the consequence of the outliers in a data set might reduce accuracy and the cost incurred by not detecting the outliers is very high. The most common type of outlier is the noise outlier and this is considered in this research work.

Approaches to detect and remove outliers have been studied by several researchers. Some techniques are developed for certain application domains, while others are more generic. These approaches can be classified into Distribution-based approaches, Depth-based approaches, Clustering-based approaches, Distance-based approaches and Density-based approaches. Each of these methods has its own advantages and disadvantages.

In general, all these methods use two steps for outlier detection. The first identifies a profile around a data set using a set of inliers (normal data). In the second step, a data instance is analyzed and identified as outlier when its attributes are different from the attributes of inliers. All these methods assume that all normal instances will be similar, while the outliers will be different.

Over the last decade of research, distance-based outlier detection algorithms have emerged as a viable, scalable, parameter-free alternative to the more traditional statistical approaches. For this purpose, the density-based LOF (Local Outlier Factor) algorithm is used with enhancements.

4.2.1. Density-Based LOF Algorithm

The density-based outlier mining algorithm is an effective approach to detect outlier points. The Local Outlier Factor (LOF) is an outlier detection algorithm (Breunig et al., 2000) where the key idea is comparing the local
density of a point’s neighborhood with the local density of its neighbors. The traditional density based algorithm is described as follows.

Assume that $C$ is the core object in dataset $D \in \mathbb{R}^d$ and $\varepsilon$ is its neighborhood radius. Given an object $o \in D$ and a number $m$, such that for every $C \in D$, if $o$ is not within the $\varepsilon$-neighborhood of $C$ and $|o_{\varepsilon\text{-set}}| \leq m$, $o$ is called the density based outlier with respect to $\varepsilon$ and $m$. Given any object $P$ in dataset $D$ and integer $m$, the density-based algorithm first computes all neighborhood objects in the $\varepsilon$-neighborhood of $P$ and judges whether it is a core object to find all outliers. If the $\varepsilon$-neighborhood of an object is within the $\varepsilon$-neighborhood of $P$ (that is core object is within $\varepsilon$-neighbourhood of $P$), then all objects whose number is more than $m$ is treated as inliers; otherwise, $P$ is considered as an outlier. The iteration does not terminate until all the objects in $D$ are processed. The detailed description of the above process is given below.

The algorithm works on the concept of a local density, where locality is given by the $k$-nearest neighbors, whose distance is used to estimate the density. By comparing the local density of an object to the local densities of its neighbors, the regions of similar density and points that have a substantially lower density than their neighbors are identified. These are considered to be outliers. This process is illustrated in Figure 4.2.

![Figure 4.2: Basic Idea of LOF](Source: www.wikipedia.org)

The local density is estimated by the typical distance at which a point can be "reached" from its neighbors. The definition of "reachability distance"
used in LOF is an additional measure to produce more stable results within clusters. Several concepts and terms are used to explain the LOF algorithm and are defined below.

**Definition 1: k-distance of an object p**

For any positive integer k, the k-distance of object p (k-distance(p)), is defined as the distance d(p,o) between p and an object of D such that:

- for at least k objects o' ∈ D\{p} it holds that d(p,o') ≤ d(p,o), and
- for at most k - 1 objects o' ∈ D\{p} it holds that d(p,o') < d(p,o).

**Definition 2: k-distance neighborhood of an object p**

Given the k-distance of p, the k-distance neighborhood of p contains every object whose distance from p is not greater than the k-distance.

**Definition 3: reachability distance of an object p with respect to object o**

Let k be a natural number. The reachability distance of object p with respect to object o is defined using Equation (4.61).

\[
\text{Reach – dist}_k(p) = \max\{k\text{-distance}(o), d(p,o)\} \quad (4.61)
\]

Let MinPts be the only parameter and the values reach-dist_{MinPts}(p, o), for o ∈ N_{MinPts}(p), be a measure of the volume to determine the density in the neighborhood of an object p. For example, the reachability distance of an object A from B is the true distance of the two objects, but at least the k-distance of B. Objects that belong to the k nearest neighbors of B are considered to be equally distant. The reason for this distance is to get more stable results. Note that this is not a distance in the mathematical definition, since it is not symmetric. An illustration of the reachability distance is shown in Figure 4.3 where the objects B and C have the same reachability distance (k=3), while D is not a k nearest neighbour. The size of the neighborhood of the
object is determined by the area containing a user-supplied minimum number of points (MinPts).

![Diagram of four points A, B, C, and D with a circle around A]

Figure 4.3: Reachability Distance

**Definition 4: Local reachability density of an object p**

The local reachability density of p is defined as

$$\text{lr}_{\text{Minpts}}(p) = \frac{1}{\sum_{o \in N_{\text{Minpts}}(p)} \text{reach} - \text{dist}_{\text{Minpts}}(p, o)}$$  \hspace{1cm} (4.62)

This is the quotient of the average reachability distance of the object A from its neighbors. The local reachability is not the average reachability of the neighbors from A (which by definition is k-distance(A)), but the distance at which it can be “reached” from its neighbors.

**Definition 5: Local outlier factor of an object p**

The local reachability densities are then compared with those of the neighbors using the LOFs. The local outlier factor of p is defined by Equation (4.63).

$$\text{LOF}_{\text{Minpts}}(p) = \frac{\sum_{o \in N_{\text{Minpts}}(p)} \text{lr}_{\text{Minpts}}(o)}{|N_{\text{Minpts}}(p)|}$$  \hspace{1cm} (4.63)
The equation produces the average local reachability density of the neighbors divided by the objects own local reachability density. A value of approximately one indicates that the object is comparable to its neighbors (and thus not an outlier). A value below one indicates a denser region (which would be an inlier), while values significantly larger than 1 indicate outlier. It is also shown that the lower \( \text{lrd}_{\text{MinPts}}(p) \) is and the higher the \( \text{lrd}_{\text{MinPts}}(o), o \in N_{\text{MinPts}}(p) \) are, the higher is the LOF value of \( p \). The outlier factor of object \( p \) captures the degree to which \( p \) can be called as an outlier.

The density based outlier detection algorithm is performed in two steps. In the first step, reachability was calculated with four inputs, namely, the Dataset \( D \), lower number of neighbours \( l_{bn} \), upper number of neighbours \( u_{bn} \), distance ‘g’ (maximum distance between center to objects). \( g_{\text{dist}} \)-neighbors means neighbour objects in ‘g’ distance. The \( \text{lrd}_{\text{objects}} \) means that lower reachability data objects, \( N \) carries number of objects and \( o \) objects. This procedure is able to generate details of objects and their mutual reachability (Figure 4.4). The important task of identifying the outliers is performed in step 2 (Figure 4.5).

The computation complexity of LOF algorithm is \( O(n^2) \), where \( n \) is the number of objects in \( D \). The main drawback here is that all objects have to be loaded onto the main memory to verify whether an object is within the neighbourhood of the given objects. In large dataset, however, the memory requirements for data storing and computing will be significant and difficult to predict and the performance of LOF quickly becomes untenable. To solve this problem, a data partition method can be used and is explained in the following section.
4.2.2. Speed Optimizers

The speed optimizers presented in this section is based on the following observation of the traditional LOF algorithm.
Let $x$ be a data point in $X$, $k$ be an integer and $t$ be a real number, the neighborhood density of a data point $x \in X$ can be defined in three ways:

\textbf{Definition 6 : Density $D(x, k, t)$}

$$D(x, k, t) = \begin{cases} 0 & \text{if there are fewer than } k \text{ data points within distance } t \\ 1 & \text{otherwise.} \end{cases}$$

The definition considers an outlier as a binary property such that the obtained density for each data point divides the whole dataset into exactly two groups: inliers and outliers. There is no measure of how much a data point is outlying and the identified outliers are not ranked.

\textbf{Definition 7 : Max Density $D_m(x, k)$}

$D_m(x, k)$ is the reciprocal of the distance to the $k$-th nearest neighbor.

\textbf{Definition 8 : Average Density $D_a(x, k)$}

$D_a(x, k)$ is the reciprocal of the average distance to the $k$ nearest neighbors.

Definitions 7 and 8 introduce the ranking mechanism based on the distance to the $k$ nearest neighbors of a point. The identified outliers are more meaningful since the information of the degree of being an outlier has been integrated into the analysis process. This concept is generalized by the LOF algorithms. They introduce an outlier degree that takes into consideration the data point’s relative density when compared with its nearest neighbours (Breunig et al., 1999). The advantage of LOF is that the local densities of the non-outlier datapoints will have less impact on the ranking of the outliers.

As noted before, the major parameter in LOF is MinPts, the minimum number of the nearest neighbors to consider. This parameter is highly application-dependent and some insight into the structure of the dataset is required in order to set it correctly. The main challenge in the selection of
MinPts is the fact that the LOF of a given data point is not monotone in MinPts. Another related issue with LOF is the existence of duplicated data in a dataset. Roughly speaking, the LOF of duplicated data points is infinity unless the MinPts is larger than the number of duplicated data points. As has been mentioned by Breunig et al. (2000), this difficulty can be overcome by slightly changing the original definition of LOF to ignore the neighboring data points that are duplicated.

Two speed optimizers are analyzed in this chapter. The first speed optimizer (Yang and Huang, 2008) is based on Definition 7, where the maximum distance is taken into account for outlier detection and the second follows definition 5 where an outlier score is used for outlier detection. The second optimizer (Pei et al., 2006) used the relative degree of neighborhood density with respect to a fixed set of reference points to approximate the degree of density defined in the distance-based method.

A) Speed Optimizer 1

Data partitioning is an effective technology to solve the problem of memory and huge computations in LOF. Due to the use of global threshold neighborhood radius and density in LOF, it is difficult to achieve a better mining result on the dataset with inconsistent distribution. So, the dataset can be partitioned into several subsets and assign different thresholds for each subset. In addition, data partitioning is conducive to the implementation of parallel computation, which will greatly increase the speed of mining. During data partitioning, the entire dataset is divided into a number of local subsets according to the sample distribution characteristics to ensure that the data distribution in local subset is as identical as possible. Then, calculate the appropriate neighborhood radius and implement LOF algorithm in each subset. By analyzing the obtained local outliers and their adjacent objects, outliers can be determined. Since the local neighborhood radius is used for outlier
detection, the poor effect brought by using global neighborhood radius will be released or even eliminated.

Let $D$ be a dataset and $C$ be a core object with respect to its neighborhood radius, $\varepsilon$ and $m$ in $D$. Let $C_{\varepsilon}$-set denote the $\varepsilon$-neighborhood set of $C$. Then for every $P \in C_{\varepsilon}$-set, $d(C, P) \leq \varepsilon$, where $d(C, P)$ is the Euclidean distance between $P$ and $C$. It means that $P$ is within the $\varepsilon$-neighborhood of $C$ and it is not an outlier. Thus, for every data object, the check whether there are core objects in the $\varepsilon$-neighborhood of it can be eliminated. If $P$ is a core object, then $P$ and all data in its’ $\varepsilon$-neighborhood will be marked as inliers. Otherwise, analysis is performed to check the existence of core objects in $\varepsilon$-neighborhood of $P$. If exist then it means that $P$ is in the $\varepsilon$-neighborhood of the core object and $P$ is not an outlier. This step reduces the computational cost considerably. The mathematical definitions used by the algorithm are given below.

**Definition 9 : Module Definition**

For any dataset $D$, the module $P = (P_1, P_2, \ldots P_d)$ and $P \in D$ is defined using Equation (4.64).

$$M(P) = \sqrt{\sum_{i=1}^{d} P_i^2}$$ (4.64)

**Definition 10 : Inner Product**

If $P (P_1, P_2, \ldots P_d)$ and $Q = (Q_1, Q_2, \ldots Q_d)$ are dimensional vectors and $P, Q \in D$, then the inner product of $P$ and $Q$ is defined using Equation (4.65).

$$(P \cdot Q) = \sum_{i=1}^{d} P_i \times Q_i$$ (4.65)

**Lemma 1 :**
Given \( P, Q \in D \), \( d(P, Q) \) denotes the Euclidean distance between \( P \) and \( Q \). Then, \( d(P, Q) \geq |M(P) - M(Q)| \).

**Proof:**

According to definition 1 and 2,

\[
M(P) \times M(Q) = \sqrt{\frac{1}{d} \sum_{i=1}^{d} P_i^2 \times \sum_{i=1}^{d} Q_i^2}
\]

\[
= \sqrt{\frac{1}{d} \sum_{i=1}^{d} (P_i \times Q_i)^2 - \sum_{i=1, j=1, i \neq j}^{d} (P_i \times Q_i)^2}
\]

\[
= \sum_{i=1}^{d} (P_i \times Q_i)
\]

\[
= (P \cdot Q)
\]

Then

\[
D(P, Q) = \sqrt{\sum_{i=1}^{d} (P_i - Q_i)^2}
\]

\[
= \sqrt{\sum_{i=1}^{d} P_i^2 - 2 \sum_{i=1}^{d} P_i \times Q_i + \sum_{i=1}^{d} Q_i^2}
\]

\[
= \sqrt{M(P)^2 - 2(P \cdot Q) + M(Q)^2}
\]

\[
= \sqrt{M(P)^2 - 2M(P) \times M(Q) + M(Q)^2}
\]

\[
= |M(P) - M(Q)|
\]

(4.65)

In the modified LOF algorithm, the modules of each data point \( P \) are calculated before outlier detection and the index of the data point and its
module information is stored in a table. This table alone is loaded onto the memory during outlier detection, thus reducing the amount of memory needed. For example, given a point Q, |M(P) - M(Q)| is calculated first, while simultaneously checking the $\varepsilon$–neighborhood number of point P. If |M(P) - M(Q)| > $\varepsilon$, the d(P, Q) > $\varepsilon$ according to Lemma 1. Thus, it can be concluded that Q is an outlier without calculating d(P, Q). This process significantly reduces the computations required and thus can work as a speed optimizer to improve the outlier detection process.

B) Speed Optimizer 2

This algorithm mainly consists of two phases, namely partitioning the dataset and detecting outliers by LOF with speedup strategy in each subset. Due to their features, outliers are always sparse and have low ratio in the entire dataset. In other words, most data are impossible outliers. As mentioned previously, the analysis whether it is a core object or it is within a $\varepsilon$–neighborhood of a core object for every data object is not performed. The steps of the algorithm are outlined in Figure 4.6.

| Step 1 | Read in module table from memory, initialize $\varepsilon$ and set variable flag to 0. |
| Step 2 | Assume that current data object being processed is P. For Q $\neq$ P, calculate |M(P) - M(Q)|. |
| Step 3 | If |M(P) - M(Q)| > $\varepsilon$, then simple select next Q. Else, goto step 4. |
| Step 4 | Calculate d(P, Q). If Q is within the $\varepsilon$–neighborhood of P, the number of P $\varepsilon$-set plus 1. |
| Step 5 | For every $C \in P\varepsilon$-set, if the flag of $C$ is 1, process is broken; otherwise, judge whether $C$ is a core object. |

Figure 4.6 : Modified LOF Algorithm
Traditional LOF algorithm need calculate the distance between data objects in order to determine that whether an object is within the $\varepsilon$--neighborhood of another. In this version, however, the module of each object in dataset is calculated in pre-processing. According to Lemma 1, if $|M(P)-M(Q)|>\varepsilon$, the distance between $P$ and $Q$ (step 2 and step 3) need not be calculated. In addition, the algorithm can effectively minimize the number seeds to identify outliers. Given any object $C$ within a $\varepsilon$-neighborhood, the flag is analyzed first. If it is a core object (flag=1), all objects within the $\varepsilon$-neighborhood of $C$ will not be outliers. Thus, $C$ is not selected as a seed to calculate the number of its $\varepsilon$--neighborhood (step 5).

The procedure to detect outliers using the modified LOF algorithm (Figure 4.6) is shown in Figure 4.7.

<table>
<thead>
<tr>
<th>Step 1 : Sample on the dataset.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 2 : Statistically analyze on all dimensionalities and choose partition points on a desirable dimensionality.</td>
</tr>
<tr>
<td>Step 3 : Divide the data into several subsets using the partition points obtained in step 2.</td>
</tr>
</tbody>
</table>

**Figure 4.7 : Outlier Detection Using Modified LOF Algorithm**

Consider the entire dataset and obtain the statistical data distribution characteristics projected on each dimensionality respectively. In the next step determine and choose partition points in such a way that it enhances the uniformity of the data density distribution. The statistical analysis of data distribution is performed using histogram method. The computation complexity of this algorithm is $O(n_1 \log n_1+...+n_k \log n_k)$, where $k$ is the number of data partitioning and $n_i (1 \leq i \leq k)$ is the number of objects in partition $i$.

**4.3. CHAPTER SUMMARY**
This chapter presented the missing value handling algorithm and outlier detection and treatment algorithm. The result after these steps is a clean, complete dataset ready for classification. The next step, using this dataset, performs customer loyalty assessment to identify loyal customers and churners. Several clustering and classification algorithms are considered for this purpose. The details of the same are presented in the next chapter, Design of Loyalty Assessment Model.