FEATURE SELECTION USING ROUGH SET BASED HEURISTIC DEPENDENCY CALCULATION

By:
MUHAMMAD SUMMAIR RAZA
2011-NUST-DirPhD-CSE-65

A thesis submitted to National University of Science and Technology for the degree of Doctor of Philosophy in the Faculty of Computer Software Engineering

Thesis Supervisor
Dr. Usman Qamar

2018

College of Electrical and Mechanical Engineering
National University of Science and Technology

Table of Contents

List of Figures and Tables

FIGURES

TABLES

Abstract

The amount of data to be processed is significantly increasing day by day. The increase in data size is not only due to more number of records but also due to substantial number of attributes added to space. The phenomenon is leading to the dilemma called curse of dimensionality i.e. datasets with exponential number of attributes. The ideal approach is to reduce the number of dimensions such that resulted reduced set contains the same information as present in the entire set of attributes. There are various approaches to perform this task of dimensionality reduction.

Recently, rough set-based approaches, which use attribute dependency to carry out feature selection, have been prominent. However, this dependency measure requires the calculation of the positive region, which is a computationally expensive task. In this research, we have proposed a new concept called the "Dependency Classes", which calculates the attribute dependency without using the positive region. Dependency classes define the change in attribute dependency as we move from one record to another. By avoiding the positive region, they can be an ideal replacement for the conventional dependency measure in feature selection algorithms, especially for large datasets. A comparison framework was devised to measure the efficiency and effectiveness of the proposed measure. Experiments on various publically available datasets show that the proposed approaches provide significant computational performance with same accuracy as provided by conventional approach. We have also recommended seven feature selection algorithms using this measure. The experimental results have shown that algorithms using the classes were more effective than their counterparts using the positive region-based approach in terms of accuracy, execution time and required runtime memory.

Chapter 1: Introduction

Knowledge is only valuable when it can be used efficiently and effectively; therefore knowledge management is increasingly being recognized as a key element in extracting its value. An example of this is Knowledge Discovery in Databases (KDD). Traditionally, data was turned into knowledge by means of manual analysis and interpretation. For many applications, this form of manual probing of data is slow, costly, and highly subjective. Indeed, as data volumes grow dramatically, this type of manual data analysis is becoming completely impractical in many domains. This motivates the need for filtering the data.

At basic level KDD comprises of five steps as shown in figure 1.1.

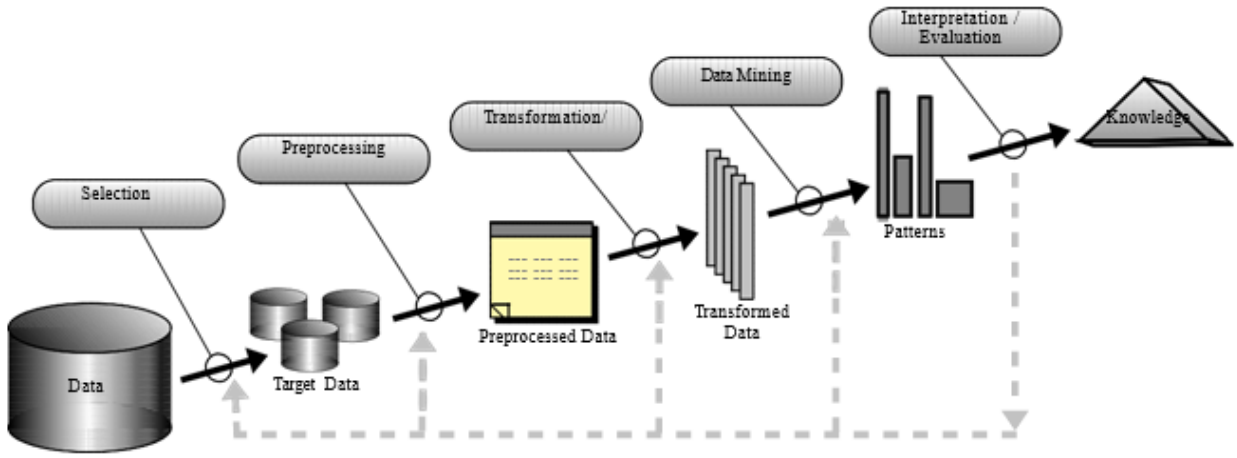


Figure-1.1: An overview of steps of KDD process taken from [2]

Here is brief description of each of the step:

- **Data Selection**

  Data selection comprises of selecting the data for knowledge discovery. This may require selecting the data from existing repository or creating a single source of data (a new repository) from multiple sources. The data is selected on the basis of the analysis task.

This is an important step where all the data relevant to analysis should be considered, failed to do so may lead to failure of the entire process.

- **Data Cleansing/ Pre-processing**

  This step refers to increasing reliability and accuracy of data. Majority of the times, the selected data may contain records that are potentially outliers, may contain insufficient details (e.g. missing attribute values), noise or incorrect values etc. Using such data may lead to incorrect models, may affect classification accuracy or performance of induction algorithms etc. at later stages. Data cleansing or pre-processing refers to removal of all such factors to enhance the quality and reliability of selected data. There are many techniques for data cleansings. We may use outlier detection algorithms to find out outliers.

- **Data Transformation/Reduction**

  This step refers to transforming the data to make it appropriate for underlying analysis and knowledge discovery. The data may contain redundant attributes that do not add much to our information or may contain totally irrelevant attributes. Such attributes are removed at this stage. Various techniques are used at this stage e.g. feature selection, feature extraction, attribute discretization etc. The basic purpose is to transform/reduce the data to enhance performance at later stages.

- **Data Mining**

  Once the data is ready we can apply our data mining algorithms to actually discover the hidden information/patterns from our data. The use of a particular mining algorithm depends on the nature of the analysis and goal of the knowledge discovery e.g. either we want prediction or description on the basis of data?

- **Interpretation/Evaluation**

  Once the knowledge has been discovered (patterns have been identified), it is evaluated on the basis of our defined goals to validate accuracy, usefulness, novelty etc. It should be noted that we may need to repeat previous steps to enhance the above mentioned measures, e.g. by including more number of features and repeating the steps again.

This research focuses on the third step i.e. data reduction of Knowledge Discovery in Datasets process. The size of a dataset comprises of two perspectives i.e. number of distinct samples to be processed per dataset and number of attributes per sample. The former only affects the training process in data mining, depending on its use, however, the latter i.e. number of attributes per sample also called dimensionality, effects training process as well as performance of algorithm. Many algorithms exhibit non polynomial execution time with respect to dimensionality.

The large number of dimensions in a dataset lead to a phenomenon called curse of dimensionality. The term was first coined by Bellman [1] resulting out of the volume increase by adding extra dimensions to mathematical space. Curse of dimensionality is the problem faced by many data analysis algorithms for their practical implementation on datasets with larger size. As already mentioned that performance of data mining algorithms is inversely proportional to dimensionality of datasets, so higher dimensionality not only challenges performance of such algorithms but makes their implementation impractical for many real life applications where datasets increase beyond smaller size.

The problems caused by curse of dimensionality lead towards finding the solution that could reduce the dimensionality without losing relevant information. There are various approaches, which can, broadly fall in two categories [2]: the ones that change or even destroy meaning of features and others that preserve semantics. Feature selection (FS) methods are semantic preserving where we select features from original on the basis of some evaluation function.

1.1 Feature Selection

Feature selection is the process of selecting a subset of features from dataset that provides most of the useful information [2]. The selected set of features can then be used on behalf of the entire dataset. So, a good feature selection algorithm should opt to select the features that tend to provide complete or most of the information as present in the entire dataset and ignore the irrelevant and misleading attributes.

The simple way to perform feature selection is to evaluate all possible subsets of features from entire dataset and evaluate their feasibility. However, exhaustive search is not possible due to its adherent implications as there will be need to evaluate 2^n subsets to be evaluated for a dataset with n features. So, exhaustive search is only possible for datasets where n is very small. An alternative way, we can use random search where a candidate feature subset can be randomly generated [3]. On each iteration, a random feature subset is selected and evaluated against its fitness for satisfaction criteria, the process repeats until we find a feature subset, at any stage, which fulfils the required criteria. The process also ends after a certain number of iterations are performed, predefined time period is elapsed or a certain number of subsets are tested.

Third and most commonly used method uses heuristic approach [3] where some heuristics function is used to guide the search. Feature selection aims at removing unnecessary features which can be classified as irrelevant features and redundant features [2]. Irrelevant features have not effect on target concept, whereas redundant features do not add any new information to target concept, instead they negatively affect the classification performance and computational time [4]. An informative feature is one having high correlation with the decision concept(s) but highly uncorrelated with other features. In the same way, a feature subset is considered to be useful if it is highly relevant and non-redundant.

In [5], authors defined two notions based on the relevancy of features: strong relevance and weak relevance; strongly relevant features are the one that cannot be removed without losing predictive accuracy. Weakly relevant features, on the other hand, may contribute to the accuracy. However, these definitions do not depend upon specific learning algorithm used.

Rough-Set Theory (RST) [6] provides a framework for dimensionality reduction. RST was proposed by Pawlak for knowledge discovery in datasets [6]. A dataset may contain number of redundant attributes can be eliminated without losing essential information. Using RST it is possible to reduce the dataset to one having lesser number of attributes but still providing maximum information. All the other attributes can be eliminated without losing information. In contrast to other co-relation based approaches minimum input is required by RST, it preserves data semantics which makes resulting models more accurate. Different algorithms have been proposed

on the basis of the concepts provided by rough set theory. Set approximation and dependency calculation are basic steps towards finding the relevant features (reducts) from the original dataset while still maintaining relevant information.

RST has been used in various domains for data analysis including economy and finance [7, 35-38], medical diagnosis [8,39-43], medical imaging [9, 44-45], banking [10,46], data mining [11, 47-50] etc.

1.2 Aims and Objectives

Rough Set Theory provides a framework comprising of data structures and operations that can be performed on these data structures for data analysis. However, one of the drawbacks of the RST is its inherent complex operations. In this research, our objective was to propose heuristics based approaches for three of the most commonly used measures of Rough Set Theory. These measures include:

- Lower approximation
- Upper approximation
- Dependency Calculation

Using the heuristics based approach will let us avoid the underlying complex operations and thus enhancing the performance and efficiency of algorithms using these measures.

We also aim at proposing feature selection algorithm using these measures which could efficiently be used to perform feature selection in case of large datasets.

The proposed heuristics measures should poses the following characteristics:

- They should provide the same accuracy as provided by the conventional Rough Set based measures.
- Performance and efficiency should be significantly enhanced, so that the algorithms using these approaches could be used for datasets beyond larger size.
- The memory requirements of the proposed approaches should be minimum as compared to the conventional approaches.

Results have shown that these all of the aims and objectives were successfully met.

1.3 Research Contribution

Rough Set Theory uses equivalence structures for calculating lower and upper approximations. The approximations are further used for performing different tasks during data analysis. Calculating equivalence class structures is computationally complex job, so in this research we have provided heuristics based approach for calculating both of these approximations. The heuristics based approach calculates these approximations without calculating equivalence class structures and thus significantly enhancing the efficiency.

Similarly, Traditional rough set based approaches use positive region based dependency measure for feature selection process. However, using positive region is computationally expensive approach that makes it inappropriate to use for large datasets. We have developed an alternate way to calculate dependency comprising of dependency classes. *A dependency class is a heuristic which defines how the dependency measure changes as we scan new records during traversal of the dataset.*

We start with first record and calculate the dependency of decision attribute on conditional attribute based on the derived heuristics. Then after adding each single record the dependency of a particular attribute is refreshed based on to which decision class, the value of that attribute leads to. On the basis of the heuristics used by dependency classes, two types of dependency classes are proposed as follows:

- Incremental Dependency Classes
- Direct Dependency Classes

Proposed Incremental Dependency Classes (IDC) are set of four classes, which govern how dependency of a decision attributes changes as a new record in dataset is added. Using these classes, lets us avoid calculation of computationally expensive positive region. Incremental dependency classes provide same accuracy as provided by conventional approach with enhanced performance.

In second phase of the research, four classes were further reduced to two without effecting the performance and accuracy. Proposed Direct Dependency Classes (DDC) are set of two classes

which categories the records in two categories. Those that are redundant on the basis of attributes considered and those which are non-redundant on the basis of attributes considered. New definitions of lower and upper approximations were also provided which are computational more efficient than traditional definitions.

On the basis of heuristics proposed, feature selection was performed by using feature selection algorithms. Positive region based dependency calculation step in these algorithms was replaced with proposed heuristics based dependency calculation. Results were compared with original ones. It was observed that proposed heuristic dependency based feature selection algorithms provide same accuracy with substantial increase in overall performance.

1.4 Structure Of The Thesis

Overall the thesis is structured as follows:

- **Chapter 2: Background**. This chapter provides overview of dimensionality reduction approaches.

- **Chapter 3: Rough Set Theory.** Chapter 3 discusses preliminary concepts of Rough Set Theory. It also provides analysis of rough-set theory along with examples including its advantages and limitations.

- **Chapter 4: Rough-sets Based Feature Selection Techniques.** This chapter discusses various feature selection techniques available in literature using Rough Set Theory.

- **Chapter 5: Dependency Classes.** In this chapter, proposed heuristics based dependency measure is discussed in detail, along with its advantages, calculation methods and analysis by comparing it with conventional rough set based dependency measure.

- **Chapter 6: Feature Selection Using Heuristics Based Dependency Classes.** This chapter discusses different feature selection algorithms using conventional dependency

measure. These algorithms are then re-implemented to use them with proposed heuristics based dependency measure.

- **Chapter 7: Results and Analysis.** This chapter discusses the results and why the proposed solution is better than the existing approaches.

- **Chapter 8: Summary and Future Work.** This section concludes the thesis. Summary of all of the findings along with overview of future work is presented.

1.5 Summary

The amount of data to be processed is significantly increasing day by day. The increase in data size is not only due to more number of records but also due to substantial number of attributes added to space. The phenomenon is leading to the dilemma called curse of dimensionality i.e. datasets with exponential number of attributes. The ideal approach is to reduce the number of dimensions such that resulted reduced set contains the same information as present in the entire set of attributes. This research addresses and tempts to solve the dilemma of curse of dimensionality by providing computationally efficient method for calculating necessary features, so that dimensions could be reduced with 0% information loss.

Chapter 2: Background

When faced with difficulties resulting from the dimensionality of a data space, one approach is to try to decrease the dimension, without losing relevant information in the data. Essentially, Dimension Reduction (DR) is used as a form of pre-processing. There are numerous methods to perform this task. An overview of techniques for dimensionality reduction is given in this chapter.

2.1 Curse of Dimensionality

The significant increase in the number of dimensions in datasets leads to phenomenon called *curse of dimensionality*. The curse of dimensionality is the problem caused by the exponential increase in volume associated with adding extra dimensions to a (mathematical) space [1]. Dimension Reduction (DR) is used as pre-processing [13]. The original feature space is mapped onto a new, reduced dimensionality space and the samples are represented in that new space [54]. There are various techniques to perform DR e.g. [55-59], but many of such techniques destroy the underlying semantics of data which makes them undesirable to many real world applications.

So, primarily this thesis focuses on DR techniques that preserve original data semantics. In particular, the research work will focus on those techniques based on Rough Set Theory [6]. Taxonomy of DR techniques is presented in figure 2.1. The presented techniques are classified into two categories: those that change the underlying semantics of data during DR process and those that preserve data semantics. The choice to choose one depends upon the underlying application, e.g. if an application needs to preserve original data semantics than the DR technique to be chosen ensure that it is preserved. However, if an application requires to discuss the relationships between attributes then the techniques that transforms the data into two or three dimensions while emphasizing these relationships may be selected.

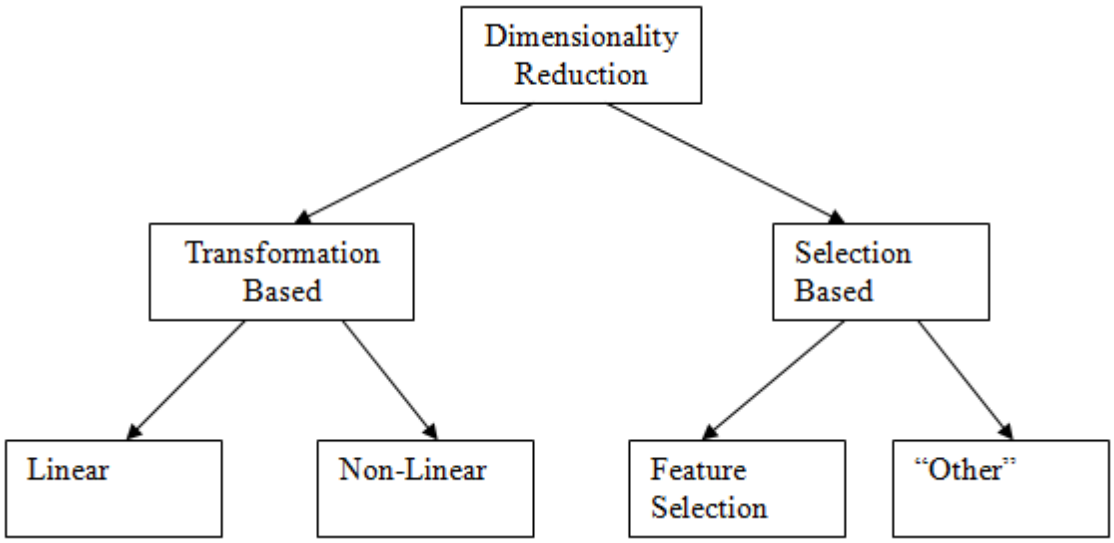


Figure 2-1: Taxonomy Of Dimensionality Reduction

Semantic preserving DR techniques "other" than feature selection have also been placed in this taxonomy. These are techniques which perform semantics-preserving dimensionality in sideline e.g. machine learning algorithm C4.5 [29]. In this chapter we will discuss sample techniques from each of the above category.

2.2 Transformation-Based Reduction

These techniques are useful where the semantics of original features are not needed by any future process. These are classified into two categories: linear and nonlinear.

2.2.1 Linear Methods

Various linear methods for DR are proposed in literature and include techniques like Principal Component Analysis [30, 31,60-62] and Multidimensional Scaling [33].

2.2.1.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) [30,31] is a well-known tool for data analysis and transformation and is considered the canonical means of DR. PCA is mathematical tool that converts large number of correlated variables to smaller number of uncorrelated variables called components. The intention is to reduce the dimensions in dataset but still preserving original

variability in data. The first principal component accounts for maximum of variability possible and each of the succeeding component accounts for maximum of remaining variability.

PCA represents variance covariance structure of high dimensional vector with few linear combinations of the original component variables. For example, for a p-dimensional random vector $\underline{X} = (X_1, X_2, ..., X_p)$, PCA will find k (univariate) random variables $Y_1, Y_2, ..., Y_k$ called K principal components and can be defined by the following formula:

$$
\begin{aligned}
Y_1 &= l_1' \underline{X} = l_{11}X_1 + l_{12}X_2 + \ldots + l_{1p}X_p \\
Y_2 &= l_2' \underline{X} = l_{21}X_1 + l_{22}X_2 + \ldots + l_{2p}X_p \\
&\vdots \\
Y_k &= l_k' \underline{X} = l_{k1}X_1 + l_{k2}X_2 + \ldots + l_{pk}X_p
\end{aligned}
\qquad (2.1)
$$

Here l_1, l_2 ,..etc coefficient vectors which are chosen on the basis of following conditions:

- First Principal Component = Linear combination $l_1'\underline{X}$ that maximizes Var($l_1'\underline{X}$) and $\| l_1 \| =1$
- Second Principal Component = Linear combination $l_2'\underline{X}$ that maximizes Var($l_2'\underline{X}$) and $\| l_2 \| =1$ and Cov($l_1'\underline{X}$, $l_2'\underline{X}$) =0
- j th Principal Component = Linear combination $l_j'\underline{X}$ that maximizes Var($l_j'\underline{X}$) and $\| l_j \| =1$ and Cov($l_k'\underline{X}$, $l_j'\underline{X}$) =0 for all k < j

It means that each principal component is a linear combination that maximizes variance of linear combination and has zero covariance with previous component.

Thus PCA maximizes the variance of datasets sample vectors along their axes by locating a new co-ordinate system and suitably transforms the samples. The new axes are constructed in decreasing order of variance such that first variable in new axes has maximum variance and so on. Correlation in new sample space is reduced or totally removed consequently resulting in reduced redundancies. Thus DR can be performed on a dataset using PCA and then selecting appropriate number of first k principal components as per requirement and discarding the rest.

PCA, however, suffers from the following shortcomings:

- It destroys the underlying semantics of data.
- It can be used only for numeric datasets
- It can only deal with linear projects and thus ignores any nonlinear structure in the data.
- Finally, human input is also required to decide how many of first principal components will be kept. Thus, the operator's task is to balance information loss against DR to suit the task at hand.

2.2.2 Nonlinear Methods

Linear DR methods are no doubt useful but their utility fails in case of nonlinear data. This motivated the development of nonlinear DR methods such as [63-68]. An example of nonlinear method is Locally Linear Embedding (LLE) [14], [15].

2.2.2.1 Locally Linear Embedding

LLE calculates reconstructions (embedding) which are low dimensional and neighbourhood preserving by using local symmetries of linear reconstructions (from high dimensional data). This can be explained better by considering the following informal analogy [15]. Initial data is three dimensional, however, taking shape of rectangular manifold (two dimensional) that has been moulded to a three dimensional S shaped curve. Now Scissors cut this manifold into small squares. Each square represents a locally linear patch of the non-linear surface. These squares are then arranged on flat surface however by preserving angular relationships between neighbouring squares. As all transformations comprise of translation, scaling or rotation only so this is a linear mapping. Through this process algorithm uses series of linear steps to find non-linear structure.

In first step, it selects neighbours in data points. This selection can be achieved using Euclidean distance for k nearest neighbours [16]. In second step, LLE computes the weights that linearly reconstruct data points using least square problem. Following cost function is used:

$$E_1(W) = \sum_{i=1}^{N} \left| (X_i - \sum_{j=1}^{\mathrm{K}} W_{ij} X_{\mathrm{Nj}}) \right|^2 \tag{2.2}$$

Finally we compute low dimensional embedded vectors by minimizing the embedded cost function:

$$E_2(Y) = \sum_{i=1}^{N} |(Y_i - \sum_{j=1}^{K} W_{ij} Y_{Nj})|^2 \qquad (2.3)$$

Figure 2.2 summarizes LLE algorithm

LLE ALGORITHM

1. Compute the neighbors of each data point, $\vec{X}_i$.
2. Compute the weights W_{ij} that best reconstruct each data point $\vec{X}_i$ from its neighbors, minimizing the cost in eq. (1) by constrained linear fits.
3. Compute the vectors $\vec{Y}_i$ best reconstructed by the weights W_{ij}, minimizing the quadratic form in eq. (2) by its bottom nonzero eigenvectors.

Figure 2-2: Summary of the LLE algorithm.

To save the time and space, LLE also tends to accumulate very sparse matrices. It avoids dynamic programming problems as well. LLE, however, does not provide any indication about how to map a test data point from input space to manifold space or how to reconstruct a data point from its low-dimensional representation.

Similar to LLE, Laplacian Eigenmaps attempts to find low-dimensional data representation while preserving local properties of the manifold [17]. In Laplacian Eigenmaps, the local properties are based on neighbours. Laplacian Eigenmaps minimizes the distance between a data point and its k nearest neighbours in an attempt to construct low low-dimensional representation of the data. Weights are used for this purpose, i.e., the distance between a datapoint and its first nearest neighbour contributes more to the cost function as compared to the distance between the datapoint and its second nearest neighbour which costs more as compared to distance between datapoint and its third neighbour and so on. Using spectral graph theory, the minimization of the cost function is defined as an Eigen problem.

2.3 Selection-Based Reduction

In contrast to transformation based techniques, which destroy the underlying semantics of data, semantics-preserving DR techniques (called feature selection) preserve original data semantics.

2.3.1 Feature Selection in Supervised Learning

In supervised learning, feature subset selection explores feature space, generates candidate subsets and evaluates/rates them on the basis of criterion which serves as guide to search process. The usefulness of a feature or feature subset is determined by both its *relevancy* and *redundancy* [2]. A feature is relevant if it determines the value of decision feature(s), otherwise it will be irrelevant. A redundant feature is the one highly correlated with other features. Thus a good feature subset is the one highly correlated with decision feature(s) but uncorrelated with each other.

The evaluation schemes used in both supervised and unsupervised feature selection techniques can generally be divided into two broad categories [2, 18]:

1. Filter approaches.
2. Wrapper methods.

2.3.2 Filter Techniques

Filter techniques perform feature selection independent of learning algorithms. Features are selected on the basis of some rank or score. A score indicating the "importance" of the term is assigned to each individual feature based on an independent evaluation criterion, such as distance measure, entropy measure, dependency measure and consistency measure [69]. Various feature filter based feature selection techniques have been proposed in literature e.g. [70-74]. In this section we will discuss some representative filter techniques along with advantages and disadvantages of each.

2.3.2.1 FOCUS

FOCUS [20] uses breadth-first search to find feature subsets that give consistent labelling of training data. It evaluates all the subsets of current size (initially one) and removes ones with least

inconsistency. The process continues until it finds a consistent subset or has evaluated all the possible subsets. Algorithm, however suffers from two major drawbacks: it is very sensitive to noise or inconsistencies in training datasets and algorithm furthermore, due to exponential growth of the features power set size, algorithm is not suitable for application in domains having large number of dimensions.

2.3.2.2 Selection Construction Ranking using Attribute Pattern (SCRAP):

SCRAP [21] performs sequential search to determine feature relevance in instance space. It attempts to identify those features that change decision boundaries in dataset by considering one object (instance) at a time, these features are considered to be most informative. Algorithm starts by selecting a random object, which is considered as first point of class change (PoC). It then selects next PoC which usually is the nearest object having different class label. After this nearest object to this having a different class label which becomes the next PoC. These two PoCs define a neighbourhood and dimensionality of decision boundary between the two classes is defined by the features that change between them. If only one feature changes between them, then it is considered to be absolutely relevant and is included in feature subset otherwise their associated relevance weights (which initially are zero), are incremented. However, if objects in the same class are closer than this new PoC and differ only by one feature then relevance weight is decremented. Objects belonging to neighbourhood are then removed and this process continues until there is no unassigned object to any neighbourhood. Final feature subset is then selected comprising of features with positive relevance weight and those that are absolutely relevant.

Major deficiency of the approach is that it regularly chooses large number of features. This normally happens in case when weights are decremented. Feature weights remain unaffected if more than one features change between a PoC and an object belonging to same class.

2.3.3 Wrapper Techniques

One of the criticism suffered by filter approaches is that the filter to select attributes is independent of the learning algorithm. To overcome this issue, wrapper approaches use classifier performance to guide the search i.e. the classifier is wrapped in the feature selection process [19].

Four popular strategies are [2, 19]:

1. Forward Selection (FS): Starting with an empty feature subset, it evaluates all features one by one, selects the best feature and combines this feature with others one by one.

2. Backward Elimination (BE): Initially it selects all features, evaluates by removing each feature one by one and continues to eliminate features until it selects the best feature subset.

3. Genetic Search applies genetic algorithm (GA) to search feature space. Each state is defined by chromosome that actually represents a feature subset. With this representation, implementation of GA for feature selection becomes quite simple. However, the evaluation of fitness function i.e. its classification accuracy, can be expensive.

4. Simulated Annealing (SA), in contrast to GA which maintains the population of chromosomes (each chromosome represents a feature subset), considers only one solution. It implements a stochastic search as there is a chance that some deterioration in solution is accepted - this allows a more effective exploration of the search space.

Forward elimination and backward elimination terminate when adding or deleting further features do not affect classification accuracy. However these greedy search strategies do not ensure best feature subset. GA and SA can be more sophisticated approaches and can be used to explore search space in a better way.

2.3.4 Unsupervised Feature Selection

Feature selection in unsupervised learning can however be challenging because the success criterion is not clearly defined. Various unsupervised feature selection techniques have been proposed in literature e.g. [83-87]. Feature selection in unsupervised learning has been classified in the same way as in supervised learning, i.e. unsupervised filters and unsupervised wrappers as discussed below.

2.3.4.1 Unsupervised Filters

The main characteristics of filter based approaches is that features are selected on the basis of some rank or score which remains independent of the classification or clustering process. Laplacian Score (LS) is one of the examples of this strategy, which can be used for DR when motivation is that the locality is preserved. The LS uses this idea for unsupervised feature selection [75]. LS selects features by preserving the distance between objects both in input and reduced output space. This criterion presumes all the features are relevant; the only thing is that they may just be redundant.

LS is calculated using a graph G that realises nearest neighbour relationships between input data points. A square matrix S is used to represent G where:

$S_{ij} = 0$ unless x_i and x_j are neighbours, in which case:

$$\mathbf{S}_{ij} = e^{-\frac{||x_i - x_j||^2}{t}}$$

Here "t" is a bandwidth parameter. L = D - S represents Laplacian of the graph and D = degree of diagonal matrix as given below

$$\mathbf{D}_{ii} = \sum_j \mathbf{S}_{ij},\ D_{ij,i\neq j} = 0 \tag{2.4}$$

LS can be calculated using following calculations:

$$\widetilde{\mathbf{m}}_i = \mathbf{m}_i - \frac{\mathbf{m}_i^\top \mathbf{D}\mathbf{1}}{\mathbf{1}^\top \mathbf{D}\mathbf{1}}\mathbf{1} \tag{2.5}$$

$$LS_i = \frac{\widetilde{\mathbf{m}}_i^\top \mathbf{L}\widetilde{\mathbf{m}}_i}{\widetilde{\mathbf{m}}_i^\top \mathbf{D}\widetilde{\mathbf{m}}_i} \tag{2.6}$$

Where m_i is the vector of values for the i^{th} feature and 1 is a vector or 1s of length n.

All the features can be scored on this criterion i.e. how efficiently they preserve locality. This idea can be appropriate for domains where locality preservation is an effective motivation [75] e.g. image analysis. However, it may not be a sensible motivation in case of irrelevant features e.g. in analysis of gene expression data or text classification etc.

2.3.4.2 Unsupervised Wrappers

Wrapper based techniques use classification or clustering process as part of feature selection to evaluate feature subsets. One such technique is proposed in [76]. Authors have used notion of a category unit (CU) [77] to present unsupervised wrapper-like feature subset selection algorithm. CU was used as evaluation function to guide the process of creating concepts and can be defined as follows:

$$CU(C,F) = \frac{1}{k}\sum_{c_l \in C}\left[\sum_{f_i \in F}\sum_{j=1}^{r_i} p(f_{ij}|C_l)^2 - \sum_{f_i \in F}\sum_{j=1}^{r_i} p(f_{ij})^2\right] \quad (2.7)$$

Here:

$C = \{C_1, \ldots\ldots C_l, \ldots\ldots C_k\}$ is the set of clusters

$F = \{F_1, \ldots\ldots F_i, \ldots\ldots F_p\}$ is the set of features.

CU calculates the difference between the conditional probability of a feature *i* having value *j* in cluster *l* and its prior probability. The inner most sum is over r feature values, the middle sum is over p features and the outer sum is over k clusters. CU is used as key concept to score the quality of clustering in a wrapper like search.

2.4 Summary

In this chapter, we have presented an overview of various dimensionality reduction techniques. In general, DR techniques can be categorized in two categories: transformation based reduction and selection-based reduction. Approaches in transformation-based category reduce dimensions in data but transform the data thus by destroying the underlying semantics. Selection based techniques, instead of transforming the dimensions, select the features, thus by preserving the data semantics. Feature selection can further be categorized into supervised feature selection and unsupervised feature selection. Various algorithms have been presented in both of these categories.

Chapter 3: Rough Set Theory

In this chapter the main aspects of Rough-Set theory (RST) are presented. The main objective of RST is to reduce data size. RST can reduce dimensionality using information contained within the dataset and, unlike other techniques mentioned in chapter 2, it also preserves the meaning of the features (i.e. it is semantics preserving). This chapter introduces the notions of *indiscernibility*, *rough set* and *reduct*, used to approximate information and to exclude redundant data.

3.1 Rough Set Theory (RST)

RST has become a topic of great interest over the past ten years and has been successfully applied to many domains by researchers. For a given dataset it is possible to find out a smaller attribute set (called reduct) that contains most of the information. So, attributes other than reduct set can be removed from dataset with minimal information loss. Pawlak has proposed RST for knowledge discovery in datasets [2, 6]. In contrast to conventional discrete sets, RST is based on the concepts of upper and lower approximations as discussed below.

In a dataset, there may be redundant attributes which may be eliminated without much of the essential information loss. Rough sets [6] let us define strong and weak relevance levels, so that redundant attributes may be removed. The concept of the reduct is fundamental in RST. Being a subset of attributes, it can distinguish all the objects in a dataset which are discernible with respect to the entire attribute set. Another important notion in RST is that of core. A core is common set of attributes in all reducts of a dataset. Both reduct and core are important concepts that are used in feature selection and dimensionality reduction. Reducts and cores are discussed in more detail in the following section.

3.1.1 Information Systems

An information system is just like a flat table or view [6] comprising of objects and their attributes. An IS (Λ) is defined by a pair (U,A) [6] as given below:

$\Lambda = (U, A)$

Here:

U = finite non empty set of objects

A = attributes of the objects

Every attribute $a \in A$ has a value set represented by V_a as shown below. Each value set of an attribute contains all possible values of that attribute.

$a : U \rightarrow V_a$

Table 3.1(a) is an information system A = *(U,A)* where:

U = {x_1, x_2, x_3, x_4, x_5, x_6, x_7}

A={Age, Incom}.

Table 3.1(a): Information System.

Customer	Age	Income
X_1	35-40	30000-40000
X_2	35-40	30000-40000
X_3	40-45	50000-60000
X_4	25-35	20000-30000
X_5	40-45	50000-60000
X_6	25-35	20000-30000
X_7	25-35	20000-30000

3.1.2 Decision Systems

Decision systems (DS) [6] are a special form of Information System having decision attribute also called the class of the object. Every object belongs to a specific class. The value of the class depends on other attributes called conditional attributes. Formally:

$\alpha = (U, C \cup \{D\})$

Where:

C = set of conditional attributes

D = Decision attribute (or class)

Table 3.1(b) shows a decision system with policy as decision attribute (or class).

Table 3.1(b): Decision System.

Customer	Age	Income	Policy
X_1	35-40	30000-40000	Platinum
X_2	35-40	30000-40000	Platinum
X_3	40-45	50000-60000	Gold
X_4	25-35	20000-30000	Silver
X_5	40-45	50000-60000	Gold
X_6	25-35	20000-30000	Silver
X_7	25-35	20000-30000	Gold

3.1.3 Indiscernibility

A decision system represents all knowledge about a model. This table may be unnecessarily large by two ways: there may be identical or indiscernible objects having more than one occurrence and there may be superfluous attributes. The notion of equivalence is recalled first. A binary relation is called equivalence relation if it is reflexive i.e. an object is in relation with itself xRx, symmetric i.e. if xRy, then yRx and transitive i.e. if xRy and yRz then xRz. The equivalence class of an element consists of all objects such that xRy.

Let $A = (U, C \cup \{D\})$ be a decision system; indiscernibility defines an equivalence relation between objects in A. For any $c \in C$ in A, there exists an indiscernibility relation $IND_A(C)$:

$$IND_A(C) = \{(O_1 = O_2) \in U^2 \mid \forall c \in C\ c(O1) = c(O2)\} \quad (3.1)$$

$IND_A(C)$ (also denoted by $[x]_c$) is called a "C-indiscernibility" relation. If two objects $(O_1,O_2) \in IND_A(C)$, then these objects are indiscernible or indistinguishable w.r.t. C. Considering the Table-3.1(a), objects x_1,x_2 are indiscernible w.r.t. attribute "Age". Similarly objects x_3 and x_5 are indiscernible w.r.t. attribute "Income".

The subscript is normally omitted if we are sure about which information system is meant. In Table-3.1(a):

IND({Age}) = {{x_1, x_2},{x_3,x_4},{x_5,x_6,x_7}}
IND({Income}) = {{x_1},{x_2},{x_3, x_4},x_5, x_6, x_7}}

3.1.4 Approximations

Most of the sets cannot be identified unambiguously, so we use approximation. For an information system where $B \subseteq A$, we can approximate the decision class X by using the information contained in B. The lower and upper approximations are defined as follows [6]:

$$X : \underline{B}X = \{x | [x]_B \subseteq X\} \tag{3.2}$$

$$X : \overline{B}X = \{x | [x]_B \cap X \neq 0\} \tag{3.3}$$

Lower approximation defines the objects that are definitely member of X with respect to information in "B". Upper approximation on the other hand contains objects that with respect to "B" can possibly be members of "X". The boundary region defines the difference between lower and upper approximation.

$$X : BN_B(X) = \overline{B}X - \underline{B}X \tag{3.4}$$

Using the decision system shown in Table 3.1(b), the situation can be sketched as Figure 3-1 below. The lower boundary of "Policy" defines all the equivalence classes that can surely belong to class Policy=Gold. Upper boundary defines classes that can possibly belong to Policy=Gold.

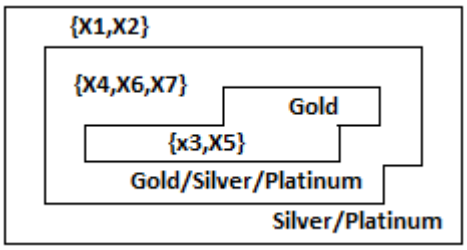


Figure 3-1: Approximation Diagram.

$\underline{B}Policy = \{\{X_3, X_5\}\}$

$\overline{B}Policy = \{\{X_3, X_5\}, \{X_4, X_6, X_7\}\}$

The boundary region is $\overline{B}Insurance - \underline{B}Insurance = \{\{X_4, X_6, X_7\}\}$. As it is non empty, so the set is rough set.

3.1.5 Positive Region

Lower approximation is also called positive region. Let P and Q be equivalence relations over U, then the positive region can be defined as:

$$POS_P(Q) = \bigcup_{X \in U/Q} \underline{P}X \tag{3.5}$$

Where P is the set of conditional attributes and Q is the Decision class. The positive region is union of all equivalence classes in $[X]_P$ that are subset of (or are contained by) target set.

Considering the table 3-1(b), we calculate positive region for set Policy="Gold" as follows:

First we will calculate $[X]_P$

Here:

$P_1 = \{x_1, x_2\}$
$P_2 = \{x_3, x5\}$
$P_3 = \{x_4, x_6, x7\}$

Now we calculate $[X]_Q$ where Q implies the concept "Insurance=Gold".

Here:

$Q = \{x_3, x_5, x_7\}$

It means we cannot distinguish between x_3, x_5 and x_7 with respect to information contained in Q.

Here for concept "Policy = Gold", only P_2 class belongs to Q. So, positive region for Q will be:

$POS_P(Q) = \{x_3, x_5\}$

3.1.6 Dependency

Dependency defines how uniquely value of an attribute determines the value of other attribute. Dependency defines how uniquely the value of an attribute determines the value of other attribute. An attribute "D" depends on other attribute "C" by degree "K" calculated by:

$$k = \gamma(C, D) = \frac{|POS_C(D)|}{|U|} \tag{3.6}$$

Where

$$POS_C(D) = \cup_{X \in U/D} \underline{C}(X) \tag{3.7}$$

is called positive region of "U/D" w.r.t. "C" as discussed in Section-3.1.5. "K" is called degree of dependency and specifies the ratio of the elements that can positively be contained by partition induced by D i.e. U/D. If K = 1, D fully depends on C, for 0 <K< 1, D depends partially on C and for K = 0, D does not depend on C. It is clear that if K=1 i.e. D totally depends on C then IND(C) ⊆ IND (D), in simple words the U/C is finer than U/D.

Finally dependency is calculated as follows:

$$k = \gamma(C, D) = \frac{|POS_C(D)|}{|U|} \tag{3.8}$$

Calculating dependency using positive region requires three steps.

1. First constructs the equivalence class structure using decision classes.
2. Construct equivalence class structure using current attribute set.
3. Calculate positive region using:

Here we provide details of each of these steps. Consider the decision system $DS = \{\{State, Qualification\} \cup \{Job\}\}$ given in Table 3.1(c):

Table 3.1(c): Sample decision system

U	State	Qualification	Job
x_1	S1	Doctorate	Yes
x_2	S1	Diploma	No
x_3	S2	Masters	No
x_4	S2	Masters	Yes
x_5	S3	Bachelors	No
x_6	S3	Bachelors	Yes
x_7	S3	Bachelors	No

We will calculate $k = \gamma(\{state, Qualification\}, Job)$ using positive region based approach.

Step-1:

First step is calculating positive region based dependency measure is to calculate equivalence classes using decision attribute ("Job" in our case):

Equivalence class structure specifies all the indiscernible objects i.e. the objects which w.r.t. to given attributes cannot be distinguished. In our case we will have two equivalence classes as follows:

Q1 = $\{x_1, x_4, x_6\}$
Q2 = $\{x_2, x_3, x_5, x_7\}$

Note that if consider the value of "Job" as "Yes" we cannot distinguish among x_1, x_4 and x_6.

Step-2:

After calculating the equivalence classes using decision attribute, next step is to calculate equivalence class structure for decision attributes (in our case "{State, Qualification"}). Calculating equivalence classes using conditional attributes requires comparison of value of each attribute for each record to find indiscernible objects. The equivalence classes in our case will be:

P1 = {x1}
P2= {x2}
P3= {x3,x4}
P4={x5,x6,x7}

Step-3:

Positive region specifies which equivalence classes in Step-2 are contained by or subset of equivalence classes identified in Step-1. First we will check which classes from P1…P4 are subset of Q1 and then we will calculate which classes from P1…P4 are subsets of Q2. This process will be used for all classes in step-2 and we will identify all classes that are subset of equivalence classes in Step-1.

Here:
$P_1 \subseteq Q_1$
$P_2 \subseteq Q_2$

No other class from P_1,P_2, P_3 and P_4 is subset of either of Q_1 and Q_2. So the dependency will be:

$$k = \gamma(C, D) = \frac{|POS_C(D)|}{|U|} = \frac{|P_1|+|P_2|}{|U|}$$

$$k = \gamma(\{State, Qualification\}, Job) = \frac{2}{7}$$

This process will take a considerable amount of time for datasets with large numbers of attributes and instances. Thus, this factor makes a positive region-based dependency measure a bad choice for use in feature selection algorithms against these datasets.

3.1.7 Reducts and Core

One way of dimensional reduction is keeping only those attributes that preserve the indiscernibility relation i.e. classification accuracy. Using selected set of attributes provides the same set of equivalence classes that can be obtained by using the entire attribute set. The remaining attributes are redundant and can be reduced without effecting classification accuracy. There are normally many subsets of such attributes called reducts. Mathematically reducts can be defined using the dependency as follows:

$\gamma(C,D) = \gamma(C',D)$ for $C' \subseteq C$ (3.9)

i.e. an attribute set $C' \subseteq C$ will be called reduct w.r.t. D, if the dependency of D on C' will be same as that of its dependency on C.

Calculating the reducts comprises of two steps. First, we calculate dependency of the decision attribute on entire dataset. Normally this is "1", however, for inconsistent datasets, this may be any value between "0" and "1". In second step we try to find the minimum set of attributes on which decision attribute has same dependency value as that of its value on entire set of attributes. In this step we may use any Rough Set based feature selection algorithm. It should be noted that there may be more than one reduct sets in a single dataset.

We will now explain it with the help of an example. Consider the table-3.2 given below.

Table 3-2: Sample decision system

U	a	b	c	D
X_1	1	1	3	x
X_2	1	2	2	y
X_3	2	1	3	x
X_4	3	3	3	y
X_5	2	2	3	z
X_6	1	1	2	x
X_7	3	3	1	y

For our first step, we calculate dependency of decision attribute "D" on conditional attributes C={a,b,c}. Here we find that:

$\gamma(C,D) = 1$

For the second step, we have to find the attribute subsets such that condition mentioned in equation (8) is satisfied. Here we see that we may have two subsets that satisfy the condition.

$\gamma(\{a,b\},D) = 1$
$\gamma(\{b,c\},D) = 1$
Representing them with R_1 and R_2:
$R_1=\{a,b\}$
$R_2=\{b,c\}$

So either of R1 and R2 provide the same classification accuracy as provided by entire of the conditional attribute set thus can be used to represent entire dataset. It is important that reduct set should be optimal i.e. it should contain minimum number of attributes to better realize its significance, however, finding optimal reduct is a difficult task as it requires exhaustive search with more number of resources. Normally exhaustive algorithms are used to find reducts in smaller datasets, however, for datasets beyond smaller size, the other category of algorithms i.e. random or heuristics based search are used, but the drawback of these algorithms is that they do not produce optimal result. So getting the optimal reducts is a trade-off between the resources and reduct size. Core is another important concept in Rough Set Theory. Normally the reduct set is not unique in a dataset i.e. we may have more than one reduct sets. Although reduct may contain the same

amount of information otherwise represented by entire attribute subset, but even in reduct there are attributes that are more important than others i.e. these attributes cannot be removed without effecting the classification accuracy of the reducts. Mathematically it can be written as $Core = \bigcap_{i=1}^{n} R_i$ where R_i is i^{th} Reduct Set. So, core is the attribute or set of attributes common to all reduct sets. In our example explained above it is clear that the attribute {b} is common in all reduct sets, so, {b} is core attribute here. Manually it can be seen that removing attribute {b} from either of the reducts effects dependency of decision class on rest of the attributes in that reduct and thus effecting the classification accuracy of the reduct.

3.2 Summary

RST provides many concepts to thoroughly analyse datasets and find irrelevant and redundant features. Given a dataset with discretized attribute values, it is possible to find a subset of the original attributes using RST that are the most informative: all other attributes can be removed from the dataset with minimum information loss. Unlike statistical correlation-reducing approaches, this requires no human input or intervention. Most importantly, it also retains the semantics of the data, which makes the resulting models more transparent to human scrutiny.

Chapter 4: Rough Set Based Feature Selection Techniques

Rough Set Theory has been successfully used for feature selection techniques. The underlying concepts provided by RST help find representative features by eliminating the redundant ones. In this chapter we will present various feature selection techniques which use RST concepts.

4.1 QuickReduct

In QuickReduct (QR) [2], authors attempt to develop a forward feature selection mechanism without exhaustively generating all possible subsets. The algorithm starts with an empty set and adds attributes one by one which result in maximum increase in degree of dependency. Algorithm continues until maximum dependency value is achieved. After adding each attribute, dependency is calculated and attribute is kept if it results in maximum increase in dependency. If at any stage the value of selected attribute set becomes equal to that of the entire dataset algorithm terminates with current selected subset as reduct. Figure-4.1 show the pseudo code of the algorithm.

QUICKREDUCT (C,D)
C, The set of all conditional features
D, The set of decision features
(1) $R \leftarrow \{\}$
(2) do
(3) $T \leftarrow R$
(4) $\forall x \in (C - R)$
(5) If $\gamma_{R \cup \{x\}}(D) > \gamma_T(D)$
(6) $T \leftarrow R \cup \{x\}$
(7) $R \leftarrow T$
(8) until $\gamma_R(D) == \gamma_C(D)$
(1) Output R

Figure-4.1:Quickreduct algorithm taken from [2]

Algorithm uses positive region based approach for calculating dependency at each step-5 and 8. However, using positive region based dependency measure requires three steps i.e. calculating equivalence classes using decision attribute, calculating dependency classes using conditional

attributes and finally calculating positive region. Using these three steps can be computationally expensive.

REVERSEREDUCT [2] is another strategy for attribute reduction, however, it uses backward elimination in contrast with forward feature selection mechanism. Algorithm starts by considering entire set of conditional attributes as reduct. It then removes one attributes at a time and calculates dependency until the removal of any further attribute becomes impossible without introducing inconsistency. Algorithm also suffers same problem as that faced by QuickReduct. It uses positive region based dependency measure and is equally unsuitable for larger datasets.

4.2 Hybrid Feature Selection Algorithm Based On Particle Swarm Optimization (PSO):

In [23] Hanna et al. presented a supervised hybrid feature selection algorithm based on Particle Swarm Optimization (PSO) and RST. Algorithm computes reducts without exhaustively generating all possible subsets. Algorithm starts with an empty set and adds attributes one by one. It constructs a population of particles with random position and velocity in S dimensions. In the problem space. It then computes fitness function of each particle using RST based dependency measure. The feature with highest dependency is selected and the combination of all other features with this one are constructed. Fitness of each of these combination is selected. If the fitness value of this particle is better than previous best (pbest) value, this is selected as pbest. Its position and fitness are stored. It then compares the fitness of current particle with population's overall previous best fitness (gbest). If it is better than gbest then gbest position is set to current the current particle's position with the global best fitness updated. This position represents the best feature subset encountered so far, and is stored in R. algorithm then updates velocity and position of each particle. It continues until stopping criteria is met which is maximum number of iterations in normal case. According to the algorithm, the dependency of each attribute subset is calculated based on the dependency on decision attribute and the best particle is chosen. Algorithm uses positive region based dependency measure and is enhancement of QuickReduct algorithm.

Velocity of each particle is represented using positive number from 1 to V_{max}. It implies that how many bits of a particle should be changed to be the same as that of global best position. The number

of bits different between two particles imply the difference between their position e.g. if Pgbest = [1,0,1,1,1,0,1,0,0,1], Xi = [0,1,0,0,1,1,0,1,0,1] then the difference between Pgbest and Xi is Pgbest − Xi=[1,−1,1,1,0,−1,1,−1,0,0]. '1' means that this bit (each "1" represents the presence of feature and "0" represents absence) should be selected as compared to the global best position and "-1" means that this bit should not be selected. After velocity is updated, next task is to update position by new velocity. If the new velocity is V, and the number of different bits between the current particle and gbest is xg, then position is updated as per following conditions:

- $V \leq xg$. In this case random V bits, which are different from that of gbest, are changed. So the particle will move towards best position while keeping its exploration ability.
- $V > xg$. In this case, apart from the bits to be the same as that of gbest, $(V - xg)$ further bits should also be randomly changed. Hence, after the particle reaches the global best position, it keeps on moving some distance toward other directions, which gives it further exploration ability.

Figure-4.2 shows the pseudo code of PSO-QR algorithm.

Input: *C, the set of all conditional features;*
D, the set of decision features.

Output: Reduct R

Step 1: Initialize X with random position and V_i with random velocity

$\forall : X_i \leftarrow$ *random Position();*

$V_i \leftarrow$ *random Velocity();*

Fit $\leftarrow 0$; *globalbest* $\leftarrow$ *Fit;*

Gbest $\leftarrow X_1$; *Pbest(1)* $\leftarrow X_1$

For i =1. . .S

pbest(i)=X_i

Fitness (i)=0

End For

Step 2 : While Fit!=1 //Stopping Criterion

For i =1. . .S //for each particle

$\forall : X_i$;

// Compute fitness of feature subset of X_i

$R \leftarrow$ *Feature subset of* X_i *(1's of* X_i*)*

$\forall x \in$ (C-R)

$$\gamma_{R \cup \{x\}}(D) = \frac{|POS_{R \cup \{x\}}(D)|}{|U|}$$

$$Fit = \gamma_{R \cup \{x\}}(D) \quad \forall x \subset R, \gamma_x(D) \neq \gamma_C(D)$$

End For

Step 3: Compute best fitness

For i=1:S

if (Fitness(i) > globalbest) // if current fitness is greater than global best fitness

globalbest $\leftarrow$ *Fitness(i); //assign current fitness value as global best fitness*

gbest $\leftarrow X_i$;

getReduct(X_i)

Exit

End if

End For

UpdateVelocity(); //Update Velocity V_i*'s of* X_i*'s*

UpdatePosition(); //Update position of X_i*'s*

//Continue with the next iteration

End {while}

Output Reduct R

Figure-4.2: PSO-QR taken from [23]

4.3 Genetic Algorithm

In [24] authors present a rough set based genetic algorithm (GA) for feature selection. The selected set of features was provided to artificial neural network classifier for further analysis. The algorithm uses positive region based dependency measure as fitness for generated candidates in proposed system. The proposed system uses RST based feature dependency value of each chromosome for finding the high performance optimal reducts. Stopping criterion was defined on the based on following equation:

$$k = \gamma(C, D) = \frac{|POS_C(D)|}{|U|} \geq \alpha$$

The candidates equal or greater than were accepted as result. Following equation was used to calculate solution addition type added to the solution:

$RSC\% = 100\% - (BSC\% + WSC\%)$

Where:

RSC = Random Selected Chromosomes

BSC = Best Solution Candidates

WSC = Worst Solution Candidates

Number of generations in each generation pool were 2*n where "n" is user defined parameter and can be changed by user for optimal performance and specifying the number of generations. In proposed version, these 2*n (2, 4, 6… n) generations were randomly initialized and used for generating the following generations. The last 2*n (4, 6, 8…) generations were used to construct the gene pool used to determine the intermediate region used for crossover and mutation operator. For crossover, order based and partially matched crossover methods were used. In order based method, random number of solution points are selected from parent chromosomes. In first chromosome selected gene will remain at its place whereas in second chromosome, the corresponding gene will be beside that of first chromosome that occupy the same place. Order based crossover method is shown in Figure-4.3 and 4.4. Figure-4.3 shows the selected chromosomes and Figure-4.4 shows the resultant chromosomes.

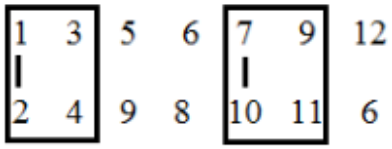


Figure-4.3: Selected Chromosomes for order based crossover method

1 2 5 6 7 10 12

2 1 9 8 10 7 6

Figure-4.4: Selected Chromosomes for order based crossover method

In partially matched method (PMX), two crossover points are randomly selected to give matching selection. Position wise exchange takes place then. It affects cross by position-by-position exchange operations. It is also called partially mapped crossover as parents are mapped to each other. Figure-4.5 and 4.6 show the process of partially mapped crossover method.

1 2 3 | 5 4 6 7 | 8 9

4 5 2 | 1 8 7 6 | 9 3

Figure-4.5: Selected Chromosomes for partial mapped crossover method

8 1 2 | 5 4 6 7 | 9 3

5 2 3 | 1 8 7 6 | 4 9

Figure-4.6: Chromosomes resulted after crossover operator

For mutation, inversion and two change mutation operators were used. In inversion method a subtour is randomly selected by determining two points in chromosome and gene are inverted between selected points where as in adjacent two input change mutation method, adjacent two genes are selected and place of genes are inverted. Figure 4.7 and 4.8 show both mutation methods:

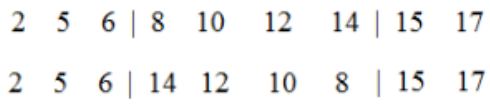


Figure-4.7: Inversion mutation method

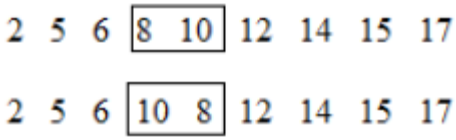


Figure-4.8: Adjacent two change mutation method

4.4 Incremental Feature Selection Algorithm (IFSA):

Qian et al. [25] present an incremental feature selection algorithm (IFSA) for feature subset selection. It starts with an original feature subset P. It then incrementally computes the new dependency function and evaluates P for either it is the required feature subset or not. If the new dependency function under P is equal to that under the whole feature set, P is also the new feature subset; otherwise, a new feature subset is computed from P. Algorithm proceeds to gradually select features with highest significance from C - P and adds them to feature subset. At final stage, algorithm removes the redundant features to ensure optimal feature subset output. Finally, redundant features are removed to ensure the optimal output in redundancy removing step.

Proposed solution compares feature significance to select the surviving features. Algorithm uses the following definitions to measure significance of an attribute:

Definition-1: Let DS = (U,A = C ∪ D) be a decision system, for B ⊆ C and a ∈ B. The significance measure of attribute "a" is defined by:

$sig_1(a,B,D) = \gamma B(D) - \gamma_{B-\{a\}}(D)$

If $sig_1(a,B,D) = 0$, then the feature "a" can be removed otherwise not.

Definition-2: Let DS = (U,A = C ∪ D) be a decision system, for B ⊆ C and a /∈ B. The significance of feature "a" is defined by:

$sig_2(a,B,D) = \gamma B \cup \{a\}(D) - \gamma B(D)$

Figure-4.9 shows the pseudo code of the algorithm.

Input A decision system $DS = (U, A = C \cup D)$, the original feature subset Red, the original positive region $\gamma_C(D)$, and the adding feature set C_{ad} or the deleting feature set C_{de}, where $C_{ad} \cap C = \varnothing$, $C_{de} \subseteq C$;
Output A new feature subset Red'.
Begin
1) Initialize $P \leftarrow Red$;
2) If a feature set C_{ad} is added into the system DS;
3) let $C' \leftarrow C \cup C_{ad}$;
4) compute the equivalence classes U/C' and $\gamma_{C'}(D)$;
//According to Theorem 1
5) for $i = 1$ to $|C_{ad}|$ do
6) compute $sig_1(c_i, C_{ad}, D)$;
7) if $sig_1(c_i, C_{ad}, D) > 0$, then $P \leftarrow P \cup \{c_i\}$;
8) end for
9) if $\gamma_P(D) = \gamma_{C'}(D)$, turn to Step 25; else turn to Step 16;
10) End if
11) If an feature set C_{de} are deleted from the system DS;
12) let $C' \leftarrow C - C_{de}$;
13) if $C_{de} \cap P = \varnothing$, turn to Step 25; else $P \leftarrow P - C_{de}$ and turn to Step 14;
14) compute the equivalence classes U/C' and $\gamma_{C'}(D)$;
//According to Theorem 2
15) End if
16) For $\forall c \in C' - P$, construct a descending sequence by $sig_2(c, P, D)$, and record the result by $\{c'_1, c'_2, ..., c'_{|C'-P|}\}$;
17) While $(\gamma_P(D) \neq \gamma_{C'}(D))$ do
18) for $j = 1$ to $|C' - P|$ do
19) select $P \leftarrow P \cup \{c'_j\}$ and compute $\gamma_P(D)$;
20) End while
21) For each $c_j \in P$ do
22) compute $sig_1(c_j, P, D)$;
23) if $sig_1(c_j, P, D) = 0$, then $P \leftarrow P - \{c_j\}$;
24) End for
25) $Red' \leftarrow P$, return Red';
End

Figure-4.9: IFSA taken from [25]

4.5 Feature Selection Method using Fish Swarm Algorithm (FSA):

Chen et al. [26] present rough set based feature selection method using fish swarm algorithm (FSA). As first step, algorithm constructs the initial swarm of fish with each fish, searching for food, represents a subset of features. With passage of time, these fish change their position to search for food, communicate with each other to find a locally and globally best position, the position with minimum high density of food. After a fish achieves maximum fitness, it perishes by getting rough set Reduct. The next iteration starts after all the fish are perished. Process continues until it gets the same reducts in three consecutive iterations or maximum iteration threshold is met. Figure-4.10 shows the flow of FSA process.

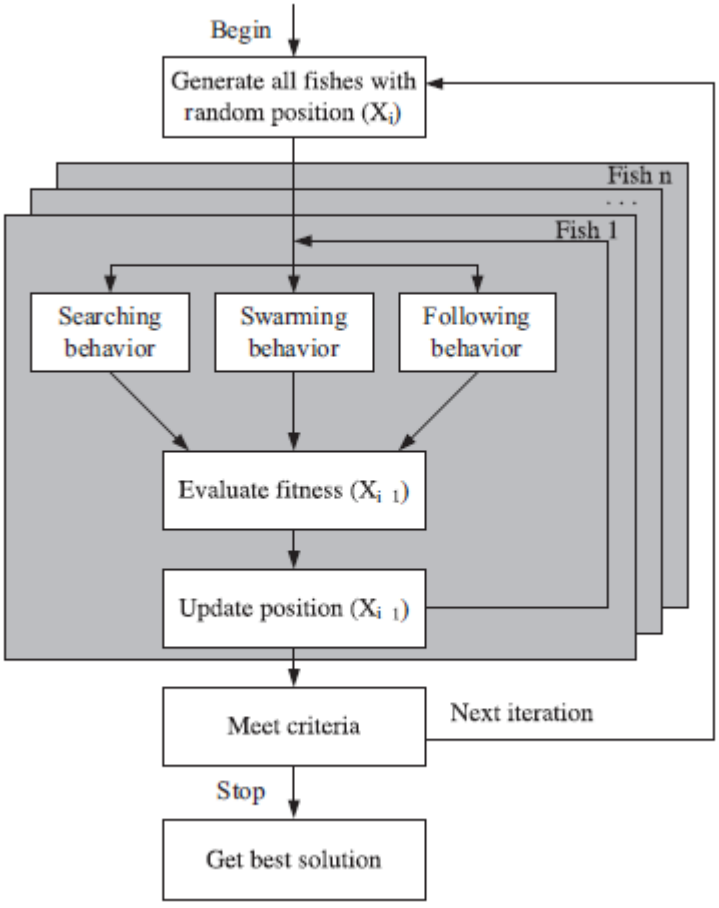


Figure-4.10: Flow of FSA

Some underlying concepts that must be considered before applying FSA to feature selection are:

4.5.1 Representation of Position:

A fish position is represented by binary bit string of length N where N is total number of features. The presence of a feature in a fish is represented by binary bit "1" and absence of a feature is represented by "0". E.g. if N=5, the following fish shown in figure-4.11 represents the presence of first, third and fourth feature from dataset

1	0	1	1	0

Figure-4.11: a sample fish

4.5.2 Distance and centre of fish:

Suppose two fish are represented by two bit strings X, Y representing the position of these two fish, hamming distance will be calculated by X XOR Y i.e. the number of bits at which strings are different. Mathematically:

$$h(X,Y) = \sum_{i=1}^{N} x_i \oplus y_i,$$

Where "$\oplus$" is modulo-2 addition, x_i; y_i $\{0, 1\}$. The variable x_i represents a binary bit in X.

4.5.3 Position Update Strategies:

In each iteration, every fish starts with a random position. Fish change their position one step according to searching, swarming and following behaviour. Authors have used fitness function to evaluate all of these behaviours. The behaviour with maximum fitness value updates the next position.

4.5.4 Fitness Function:

Following fitness function was used in the algorithm:

$$\text{Fitness} = \alpha * \gamma_R(D) + \beta * \frac{|C| - |R|}{|C|},$$

Where $\gamma_R(D)$ is dependency of decision attribute "D" on "R" and R is number of "1" bits in a fish and |C| is the number of features in dataset.

4.5.5 Halting Condition:

When a fish achieves maximum fitness it is perished by getting rough set reduct. Next iteration starts after all the fish are perished. Algorithm stops when maximum iteration threshold is met or same feature reduct is obtained under three consecutive iterations.

4.6 Feature Selection Method Based on QuickReduct and Improved Harmony Search Algorithm (RS-IHS-QR):

Inbarani et al. [27] propose a feature selection method based on QuickReduct and improved harmony search algorithm (RS-IHS-QR). This algorithm emulates the music improvisation process where each musician improvises their instrument's pitch by searching for a perfect state of harmony. The algorithm stops when it reaches the maximum number of iterations or finds a harmony vector with maximum fitness. It uses rough set based dependency measure as its objective function to measure the fitness of harmony vector which again is a performance bottleneck for larger datasets.

Table-4.1 shows the summary of all the rough set based approaches discussed so far.

Table-4.1: Related algorithms based on RST

Algorithm	Technique used	Advantages	Disadvantages
Supervised hybrid feature selection based on PSO and rough sets for medical diagnosis [22]	Particle swarm optimization and rough set based dependency measure.	PSO is an advance heuristics based algorithm to avoid exhaustive search	Conventional dependency based measure is a performance bottleneck.
Rough set based genetic algorithm [24].	Conventional positive region based dependency calculation.	It is Based on randomness so the procedure may find reducts in few attempts.	Uses conventional positive region based dependency measure.
Quick Reduct approach for feature selection [23].	Rough set based dependency measure.	Attempts to calculate reducts without exhaustively generating all possible subsets.	Uses conventional dependency based measure, which is time consuming task.
ResverseReduct [23].	Rough set based dependency measure.	Backward elimination is utilized without exhaustively generating all possible combinations.	Dependency is calculated using conventional positive region based approach.
An Incremental Algorithm to Feature Selection in Decision Systems with the Variation of Feature Set [25]	Incremental feature selection using rough set based significance measure of attributes.	Presents Feature selection for dynamic systems where the datasets keep on increasing with time.	Conventional dependency measure is used to measure attribute significance. Measuring significance requires measuring dependency twice, once with attribute and then without attribute.
Fish Swarm Algorithm [26]	Rough set based dependency used with fish swarm method for feature selection.	Attempts to find rough set reducts using the swarm logic where swarms can discover best combination of features.	Conventional dependency based measure is again a performance bottleneck.
Rough Set Improved Harmony Search Quick Reduct [27]	Rough set based dependency measure used with harmony search algorithm for feature selection	Integrates rough set theory with "improved harmony search" based alsorithm with QuickReduct for feature selection.	Uses conventional dependency based measure, which is time consuming.

4.7 Alternative to Positive Region based Methods

Many approaches have been proposed in literature to overcome computationally expensive task of calculating positive region. In this section we present few representative of these approaches.

In [28], Yu et al. proposed a “Compact Discernibility Information tree” also called CDI-tree for attribute reduction; The CDI-tree can map all nonempty elements to same path can allow them to share same prefix, which is recognized as a compact structure for storing non-empty elements in discernibility matrix. A heuristic algorithm, based on CDI-tree, is also proposed. As an approximate strategy, algorithm deletes the least important attribute in each iteration. The task is performed to ensure that only important attributes are kept by algorithm. At the same time, the algorithm uses to delete all paths including core node in each of iterations as well. However, their approach reduces execution time only by 44.14% as compared to its counterpart.

In [32], In-Kyoo et al. propose information-theoretic dependency roughness (ITDR). ITDR considers information-theoretic attributes dependencies degree of categorical-valued information systems. A new algorithm minimum-minimum roughness (MMR) based on rough set theory was also proposed. However, the execution time required for calculating ITDR is not provided.

In [29], Yuhua et al. propose an accelerator approach called forward approximation which can be used to accelerate algorithms of heuristic attribute reduction. Based on this framework they have also presented an improved heuristic feature selection algorithm (FSPA). Through the use of the accelerator, few heuristic fuzzy-rough feature selection algorithms have also been enhanced. However, their approach reduced the execution time of fuzzy positive region reduction approach by almost 50.4% and that of Fuzzy condition entropy based approach by almost 45% as compared to positive region.

In [30], Anhui et al. present a matrix based algorithm for calculating positive region, they have proposed the minimal and maximal descriptions in a covering decision system which can be easily obtained by the matrix-based methods. These descriptions are then employed to construct a new discernibility matrix. It is also pointed that use of minimal and maximal descriptions, the total number of nonempty discernibility sets in discernibility matrix can be reduced. However the execution time comparison of calculating matrix based positive region with that using original rough set method is not provided.

In [31] Essam presents an improved harmony search algorithm where feature selection was performed using discernibility matrix and fuzzy lower approximation. A special fitness function was defined fusing the classical ranking methods with the fuzzy-rough method, and applying binary operations to speed up implementation. However, the proposed approach reduces the execution time by almost 51%. The above discussed approaches are summarized in table-4.2 given below:

Table-4.2: summary of approaches to reduce the execution time of positive region

Algorithm	Technique used	Results
Minimal attribute reduction with rough set based on compactness discernibility information tree [15]	Discernibility information tree	For twelve datasets, the algorithm reduced execution time only by 44.14%.
Fuzzy-rough feature selection accelerator [27]	Forward approximation	For six datasets, approach reduced the execution time of fuzzy positive region reduction approach by almost 50.4% and that of Fuzzy condition entropy based approach by almost 45%.
Matrix-based set approximations and reductions in covering decision information systems [31]	Matrix-based	Results not compared with that of original method
Finding a Fuzzy Rough Set Reduct Using an Improved Harmony Search [7]	Discernibility matrix	For eighteen datasets, execution time is reduced by almost 51%.
Rough set approach for clustering categorical data using information-theoretic dependency measure [23]	Information-theoretic dependency	Execution time of calculating ITDR is not provided

4.8 Summary

In this section we have presented feature selection algorithms using rough set based positive region and alternate ones. Positive region based approaches use conventional dependency measure comprising of three steps to measure the fitness of an attribute for being selected for reduct set. However, using positive region is computationally expensive approach that makes these approaches inappropriate to use for larger datasets. Alternate approaches are the one that don't use positive region. However, application of such approaches has only been tested against smaller datasets which raises question for their appropriateness for larger datasets.

Chapter 5: Dependency Classes

Majority of rough set based feature selection approaches use positive region based dependency measure as underlying criteria for feature selection. However, the problem with this approach is calculation of positive region is a computationally expensive task. In this section we will discuss the proposed dependency classes to show how they can overcame the limitations of positive region.

5.1 Proposed Dependency Classes

Calculating the dependency of an attribute D on C requires scanning of the dataset and calculating the positive region of D w.r.t. C, which is a time consuming job. We have developed an alternate way to calculate dependency comprising of dependency classes. *A dependency class is a heuristic which defines how the dependency measure changes as we scan new records during traversal of the dataset.*

We start with first record and calculate the dependency of decision attribute on conditional attribute based on the derived heuristics. Then after adding each single record the dependency of a particular attribute is refreshed based on to which decision class, the value of that attribute leads to. On the basis of the heuristics used by dependency classes, two types of dependency classes are proposed as follows:

- Incremental dependency classes
- Direct dependency classes

5.1.1 Incremental Dependency Classes (IDC)

Incremental dependency classes comprise of four rules where each rule defines a class that governs how dependency of decision attribute “D” on “C” changes as we read each new record.

We will explain each incremental dependency class with the help of example. Consider the following decision system shown in Table-5.1 taken from [31]:

Table-5.1: Decision System example

U	a'	b'	c'	d'	D
A	M	L	3	M	1
B	M	L	1	H	1
C	L	L	1	M	1
D	L	R	3	M	2
E	M	R	2	M	2
F	L	R	3	L	3
G	H	R	3	L	3
H	H	N	3	L	3
I	H	N	2	H	2
J	H	N	2	H	1

Here:

C = {a', b', c', d'}

D = {D} and |U| = 10

Initially we start with the |U| = 6 where U = {a, b, d, e, f, g}. We calculate the dependency of "D" on all attributes present in C (given at the end of each column in Table-5.2).

Table-5.2: Decision System example

U	a'	b'	c'	d'	D
A	M	L	3	M	1
B	M	L	1	H	1
D	L	R	3	M	2
E	M	R	2	M	2
F	L	R	3	L	3
G	H	R	3	L	3
	0.16667	**0.3333**	**0.3333**	**0.5**	

Now we define different classes through which the dependency can be calculated, after adding a new record.

5.1.1.1 Existing Boundary Region Class

For an attribute a', if same value of attribute leads to different decision classes for example, in table-5.3, a'(L)->2,3 (i.e. the value "L" leads to decision class "2" and "3") then adding a new record with same value of a' decreases the dependency of decision on that attribute. Adding a row in this case will simply decreases the dependency.

For example, in Table-5.3,

$\gamma(a', D) = \frac{1}{6}$

After adding new record i.e. object "C", the new dataset with new dependency values are shown in Table-5.3

Table-5.3: adding new object "C"

U	a'	b'	c'	d'	D
A	M	L	3	M	1
B	M	L	1	H	1
C	L	L	1	M	1
D	L	R	3	M	2
E	M	R	2	M	2
F	L	R	3	L	3
G	H	R	3	L	3
	0.14286	**0.4286**	**0.4286**	**0.4286**	

Before adding new record: a'(L)-> 2,3 (in Table-5.3)

After adding new record: a'(L)-> 1,2,3 (in Table-5.4)

So by adding new record, the value "L" of attribute "a'" which initially was leading to two decision classes, now leads to three decision classes, so γ(a',D) becomes:

$\gamma(a', D) = \frac{1}{7}$

5.1.1.2 Positive Region Class

For an attribute a', if adding a record, does not lead to a different decision class for same value of that attribute, then dependency will increase.

For example in Table-5.3, b'(L)->1. Previous dependency value was 2/6. After adding new row (Object "C" as shown in table-5.4), b'(L)->1 sustains i.e. the value "L" of attribute b' uniquely identifies the decision class, so the new dependency will be:

$\gamma(b', D) = \frac{3}{7}$

5.1.1.3 Initial Positive Region Class

For an attribute a', if the value appears in the data set for the first time for that attribute, then dependency increases:

For example, adding new record (object “I”) as shown in Table-5.4:

Table-5.4: adding new object “I”

U	a'	b'	c'	d'	D
A	M	L	3	M	1
B	M	L	1	H	1
C	L	L	1	M	1
D	L	R	3	M	2
E	M	R	2	M	2
I	H	N	2	H	2
F	L	R	3	L	3
G	H	R	3	L	3
	0/8	**0.5**	**0.5**	**0.25**	

b’(N)->2. Initially the value “N” for b’ attribute was not present. Now adding the record for this value of b’ leads to new dependency value as follows:

$\gamma(\mathrm{b}, \mathrm{D}) = \frac{4}{8}$

5.1.1.4 Boundary Region Class

For an attribute a', if same value (which was leading to unique decision previously) of attribute leads to different decision, then adding the new record reduces the dependency.

For example adding a record “H” in Table-5.5:

Table-5.5: adding new object “H”

U	a'	b'	c'	d'	D
A	M	L	3	M	1
B	M	L	1	H	1
C	L	L	1	M	1
D	L	R	3	M	2
E	M	R	2	M	2
I	H	N	2	H	2
F	L	R	3	L	3
G	H	R	3	L	3
H	H	L	3	L	3
	0	**1/9**	**0.5**	**0.25**	

The new dependency γ(b’,D) will be:

$\gamma(c, D) = \frac{1}{9}$

Which was $\gamma(c, D) = \frac{3}{9}$ before adding record "H".

Table-5.6 shows the summary of all decision classes:

Table-5.6: Summary of IDC

Decision class	Definition	Initial attribute value	After adding new record	Effect on dependency
Existing Boundary region class	If same value of attribute leads to different decision classes, it decreases the dependency	a'(L)->2,3	a'(L)-> 1,2,3	Decreases
Positive region class	If adding a record, does not lead to a different decision class for same value of that attribute, then dependency will increase.	b'(L)->1	b'(L)->1	Increases
Initial Positive region class	If the value appears in the data set for the first time for that attribute, then dependency increases.	-	b'(N)->2	Increases
Boundary region class	If same value (which was leading to unique decision previously) of attribute leads to different decision, then adding the new record reduces the dependency	b'(L)->1	b'(L)->1,3	Decreases

5.1.1.5 Mathematical representation of IDC

Now we provide mathematical representation of IDC and an example about how to calculate IDC. Mathematically:

$$\gamma(\{att\}, D) = \frac{1}{N}\sum_{k=1}^{N}\gamma'_k \quad (5.1)$$

Where:

$$\gamma'_k = \begin{bmatrix} 1 & if\ V_{\{att\},k}\ leades\ to\ a\ positive\ region\ class \\ 1 & if\ V_{\{att\},k}\ leades\ to\ an\ initial\ positive\ region\ class \\ -n & if\ V_{\{att\},k}\ leades\ to\ a\ boundary\ region\ class \\ 0 & if\ V_{\{att\},k}\ has\ already\ lead\ to\ boundary\ \ region\ class \end{bmatrix}$$

Where:

$\gamma(\{att\}, D)$ *is* dependency of attribute "D" on attribute {att}

{att} is current attribute under consideration

D is decision attribute (Decision Class)

γ'_k is dependency value contributed by object "k" for attribute {att}

V{att},k is value of attribute {att} for object "k" in dataset

n is total number of previous occurrences of V{att},k

N is total number of records in dataset

5.1.1.6 Example:

Following example shows how IDC calculates dependency. We read each record and identify its dependency class. Based on the class we decide the factor by which dependency will be added in overall dependency value. We will consider dataset given in table-5.2. Using IDC:

$$\gamma(\{att\}, D) = \frac{1}{N}\sum_{k=1}^{N} \gamma'_k$$

Consider the attribute {a'}, we read first three records i.e. object "A", as it appears for the first time, so it belongs to "Initial Positive Region" class, thus we will add "1" to overall dependency value. Reading objects "B" and "C" lead to "Positive region" class, so we will add "1" for both. Reading "D", the value "L" now leads to decision class "2" (previously its one occurrence was leading to decision class "1"), so it belongs to "Boundary region" class and thus we will add the value "-1" to overall dependency. Reading object "E" at this stage, value "M" belongs to "Boundary region" class and it had two occurrences before, so, we will add "-2" in overall dependency. Reading object "F", note that it has already lead to "Boundary region" class, so we will add "0" to overall dependency and so on.

$$\gamma(\{a'\}, D) = \frac{1}{10}\sum_{k=1}^{10} \gamma'_k = (1 + 1 + 1 + (-1) + (-2) + 0 + 1 + 1 + (-2) + 0) = \frac{0}{10} = 0$$

Similarly dependency of "D" on {b', c', d'} will be as follows:

$$\gamma(\{b'\}, D) = \frac{1}{10}\sum_{k=1}^{10} \gamma'_k = (1 + 1 + 1 + 1 + 1 + (-2) + 0 + 1 + (-1) + 0) = \frac{3}{10}$$

$$\gamma(\{c'\}, D) = \frac{1}{10}\sum_{k=1}^{10} \gamma'_k = (1 + 1 + 1 + (-1) + 1 + 0 + 0 + 0 + 1 + (-2)) = \frac{2}{10}$$

$$\gamma(\{d'\}, D) = \frac{1}{10}\sum_{k=1}^{10} \gamma'_k = (1+1+1+(-2)+0+1+1+1+(-1)+0) = \frac{3}{10}$$

Note that if a value of an attribute has already lead to boundary region class than adding a same value will simply be reflected by adding "0" as dependency.

5.1.2 Direct Dependency Classes (DDC)

Direct dependency classes are alternate to IDC for calculating dependency directly without involving positive region and exhibit almost same performance as shown by IDC. DDC determines the number of unique/non-unique classes in a dataset for an attribute C. *A unique class represents the attribute values that lead to unique decision class throughout dataset, so this value can be used to precisely define decision class.*

For example in Table-5.3 the value "L" of attribute b' is unique class as all of its occurrences in the same table lead to a single/unique decision class (i.e. "1"). *Non-unique classes represent the attribute values that lead to different decision classes, so they cannot be precisely used to determine the decision class.* For example in Table-5.3, the value "R" of attribute b' represents non-unique class as some of its occurrences lead to decision class "2" and some occurrences lead to decision class "3".

Calculating unique/non-unique classes directly lets us avoid complex computations of positive region. The basic idea behind the proposed approach is that number of unique classes increase dependency and non-unique classes decrease dependency. For a decision class D, the dependency K of D on C is shown in Table-5.7.

Table-5.7: How DDC calculates dependency

Dependency	No of unique/non-unique classes
0	If there is no unique class
1	If there is no non-unique class
n	Otherwise where 0 < n < 1

The dependency using DDC approach can be calculated by the following formula:

If we consider the number of unique classes:

$$\gamma(\{att\}, D) = \frac{1}{N}\sum_{i=1}^{m}(1) \quad (5.2)$$

If we consider non-unique classes:

$$\gamma(\{att\}, D) = 1 - \frac{1}{N}\sum_{i=1}^{n}(1) \quad (5.3)$$

Where:

$\gamma(\{att\}, D)\ is\ d$ependency of attribute "D" on attribute {att}

{att} is current attribute under consideration

D is decision attribute (Decision class)

m is total number of values leading to unique decision classes

n is total number of values leading to non-unique decision classes

N is total number of records in dataset

5.1.1 Example

We consider the decision system given in table-5.2.

As per definitions of unique dependency classes:

$$(\{att\}, D) = \frac{1}{N}\sum_{i=1}^{m}(1)$$

If consider attribute {b'}, there are three unique dependency classes i.e. there are three occurrences of value "L" that lead to unique decision class, so:

$$\gamma(\{b'\}, D) = \frac{1}{10}\sum_{i=1}^{3}(1)$$

$$\gamma(\{b'\}, D) = \frac{1}{10}(1+1+1) = \frac{3}{10}$$

Similarly for attribute {c'}:

$$\gamma(\{c'\}, D) = \frac{1}{10}\sum_{i=1}^{2}(1)$$

$$\gamma(\{c'\}, D) = \frac{1}{10}(1+1) = \frac{2}{10}$$

On the other hand if we consider non-unique dependency classes:

$$\gamma(\{att\}, D) = 1 - \frac{1}{N}\sum_{i=1}^{n}(1)$$

If we consider attribute "b'", note that there are seven non-unique dependency classes (four occurrences of value "R" lead to two decision classes and three occurrences of value "N" lead to three decision classes), so:

$$\gamma(\{b'\}, D) = 1 - \frac{1}{10}\sum_{i=1}^{7}(1) = 1 - \frac{1}{10}(1+1+1+1+1+1+1) = \frac{3}{10}$$

Similarly:

$$\gamma(\{c'\}, D) = 1 - \frac{1}{10}\sum_{i=1}^{8}(1) = 1 - \frac{1}{10}(1+1+1+1+1+1+1+1+1) = \frac{2}{10}$$

Note that there are three unique decision classes in attribute {b'} and seven in {c'}.

For a decision system:

No of unique classes + no of nonunique classes = size of universe

So we either need to calculate number of unique classes or non-unique classes.

The algorithm for DDC is shown in Figure-5.1.

```
Function FindNonUniqueDependency
Begin
InsertInGrid(X1)
For i=2 to TotalUnivesieSize
    IfAlreadyExistsInGrid(Xi)
         Index = FindIndexInGrid(Xi)
         If DecisionClassMatched(index,i) =   False
           UpdateUniquenessStaus(index)
         End-IF
   Else
          InsertInGrid(Xi)
  End-IF

Dep=0

For i=1 to TotalRecordsInGrid
  If Grid(I,CLASSSTATUS)  = 1
    Dep= Dep+ Grid(i,INSTANCECOUNT)
 End-IF
Return (1-Dep)/TotalRecords
End Function

Function InsertInGrid(Xi)
For j=1 to TotalAttributesInX
  Grid(NextRow,j) = Xij
End-For
  Grid(NextRow,DECISIONCLASS) = Di
  Grid(NextRow, INSTANCECOUNT) = 1
  Grid(NextRow, CLASSSTATUS) = 1 // 1 => uniqueness
```

```
Function FindUniqueDependency
Begin
InsertInGrid(X1)
For i=2 to TotalUnivesieSize
    IfAlreadyExistsInGrid(Xi)
         Index = FindIndexInGrid(Xi)
         If DecisionClassMatched(index,i) =   True
           UpdateUniquenessStaus(index)
         End-IF
   Else
          InsertInGrid(Xi)
  End-IF

Dep=0

For i=1 to TotalRecordsInGrid
  If Grid(i,CLASSSTATUS)  = 0
    Dep= Dep+ Grid(i,INSTANCECOUNT)
 End-IF
Return Dep/TotalRecords
End Function

Function InsertInGrid(Xi)
For j=1 to TotalAttributesInX
  Grid(NextRow,j) = Xij
End-For
  Grid(NextRow,DECISIONCLASS) = Di
  Grid(NextRow, INSTANCECOUNT) = 1
  Grid(NextRow, CLASSSTATUS) = 1 // 1 => uniqueness
```

```
End-Function

Function IfAlreadyExistsInGrid(Xi)
  For i=1 to TotalRecordsInGrid
     For j=1 to TotalAttributesInX
      If Grid(i,j) <> Xj
            Return False
     End-For
  End-For
Return True
End-Function

Function FindIndexInGrid(Xi)
  For i=1 to TotalRecordsInGrid
     RecordMatched=TRUE
     For j=1 to TotalAttributesInX
      If Grid(i,j) <> Xj
            RecordMatched=FALSE
     End-For
   If RecordMatched= TRUE
     Return j
   End-If
  End-For
Return True
End-Function

Function DecisionClassMatched(index,i)
    If Grid(index, DECISIONCLASS) = Di
      Return TRUE
    Else
      Return False
   End-If
End-Function

Function UpdateUniquenessStaus(index)
   Grid(index, CLASSSTATUS) = 1
End-Function
```

DDC using non-unique classes

```
End-Function

Function IfAlreadyExistsInGrid(Xi)
  For i=1 to TotalRecordsInGrid
     For j=1 to TotalAttributesInX
      If Grid(i,j) <> Xj
            Return False
     End-For
  End-For
Return True
End-Function

Function FindIndexInGrid(Xi)
  For i=1 to TotalRecordsInGrid
     RecordMatched=TRUE
     For j=1 to TotalAttributesInX
      If Grid(i,j) <> Xj
            RecordMatched=FALSE
     End-For
   If RecordMatched= TRUE
     Return j
   End-If
  End-For
Return True
End-Function

Function DecisionClassMatched(index,i)
    If Grid(index, DECISIONCLASS) = Di
      Return TRUE
    Else
      Return False
   End-If
End-Function

Function UpdateUniquenessStaus(index)
   Grid(index, CLASSSTATUS) = 1
End-Function
```

DDC using Unique Classes

Figure-5.1: Pseudo code for directly finding dependency using unique and non-unique classes.

Grid is the main data structure used to calculate dependency directly without using positive region. It is a matrix with following dimensions:

No. of rows = No. of records in dataset

No. of Columns = number of conditional attributes + number of decision attributes + 2

So if there are ten records in dataset, five conditional attributes and one decision class then grid dimension will be 10x8 i.e. ten rows and eight columns. A row read from the dataset will first be stored in grid if it does not already exist. The five conditional attributes will be stored in first five columns; decision attribute will be stored in sixth column called DECISIONCLASS. Seventh column called INSTANCECOUNT will store the number of times that record appears in actual dataset, finally the last column called CLASSSTATUS will store the uniqueness of the record, the

value “0” mean record is unique and “1” means it is non-unique. If a record, read from dataset already exists in Grid, its INSTANCECOUNT will be incremented. If the decision class of the record is different from that already stored in Grid i.e. the same values of attributes now lead to different decision class, CLASSSTATUS will be set to “1”. However, if the record is inserted for the first time, INSTANCECOUNT is set to “1” and CLASSSTATUS is set to 0 i.e. it is considered unique.

The functions “FindNonUniqueDependency” and “FindUniqueDependency” are main functions to calculate the dependency. Functions insert the first record in Grid and then search for the same record in the entire dataset. The status of the record is updated in Grid as soon as further occurrences of the same record are found. Finally the functions calculate dependency value on the basis of uniqueness/non-uniqueness of classes. Function “InsertInGrid” inserts the record in the next row of the Grid. “FindIndex” finds the row no. of the current record in the Gird. “IfAlreadyExistsInGrid” finds if the record already exists or not. Finally “UpdateUniquenessStaus” function updates the status of the record in Grid.

5.2 Redefined Approximations

Unique decision classes lead to the idea of calculating lower and upper approximation without using indiscernibility relation. Calculating lower and upper approximation requires calculating equivalence classes (indiscernibility relation) which is computationally an expensive task. Using unique decision classes lets us avoid this task and we can directly calculate lower approximation. The new definitions are semantically same to the conventional definitions but provide computationally more efficient method for calculating these approximations by avoiding equivalence class structure. The following section discusses the proposed new definitions in detail.

5.2.1 Redefined Lower Approximation

The conventional rough set based lower approximation defines the set of objects that can with certainty be classified as members of concept “X”. For attribute(s) $c \in C$ and concept X, the lower approximation will be:

$\underline{C}X = \{X | [X]_c \subseteq X\}$

This definition requires calculation of indiscernibility relation i.e. equivalence class structure $[X]_c$, where the objects belonging to one equivalence class are indiscernible with respect to the information present in attribute(s) $c \in C$.

Based on the concept of lower approximation provided by RST, we have proposed a new definition as follows:

$$\underline{C}X = \{\forall\, x \in U, c \in C,\ a \neq b |\ x_{\{c \cup d\}} \rightarrow a,\ x_{\{c \cup d\}} \nrightarrow b\} \quad (5.2)$$

i.e. the lower approximation of concept "X" w.r.t. the attribute set "c", is set of objects such that for each occurrence of the object, the same value of conditional attribute "c" always leads to the same decision class value. So, if there are "n" occurrences of an object, then all of them lead to same decision class (for same value of attributes), which alternatively means that for a specific value of an attribute, we can with surety say that object belongs to a certain decision class. This is exactly equal to conventional definition of lower approximation.

So, the proposed definition is semantically same as conventional one, however, computationally it is more convenient for calculating lower approximation, it avoids construction of equivalence class structures to find the objects belonging to positive region. It directly scans the dataset and finds the objects that lead to same decision class throughout thus enhancing the performance of algorithm using this measure.

We will use the table-5.8 as sample to calculate lower approximation using both definitions.

Table-5.8: Sample decision system

U	a	B	c	d	Z
X1	L	3	M	H	1
X2	M	1	H	M	1
X3	M	1	M	M	1
X4	H	3	M	M	2
X5	M	2	M	H	2
X6	L	2	H	L	2
X7	L	3	L	H	3
X8	L	3	L	L	3
X9	M	3	L	M	3
X10	L	2	H	H	2

We suppose the concept: $X = \{x \mid Z(x) = 2\}$ i.e. we will find the objects about which could with surety say that they lead to decision class "2".

Conventional definition requires three steps given below:

Step-1: Calculate the objects belonging to the concept X. Here is our case concept $X = \{x \mid Z(x) = 2\}$

So, we will identify the objects belonging to the concept. In our case:

$X = \{X4, X5, X6, X10\}$

Step-2: Calculating equivalence classes using conditional attributes.

Here we will consider only one attribute "b" for simplicity i.e. $C = \{b\}$. So, we will calculate equivalence classes using attribute "b", which in our case becomes:

$[X]_c = \{X1, X4, X7, X8, X9\}\{X2, X3\}\{X5, X6, X10\}$

Step-3: Find objects belonging to lower approximation

In this step we actually find the objects that belong to lower approximation of the concept w.r.t. to considered attribute. Mathematically, this step involves finding objects from $[X]_c$ which are subset of X i.e. $\{[X]_c \subseteq X\}$

In our case:

$\underline{C}X = \{[X]_c \subseteq X\} = \{X5, X6, X10\}$

Using the proposed definition, we construct the lower approximation without using equivalence class structures. We directly find the objects that under the given concept always lead to same decision class (concept) for the current value of attributes.

In our case, we just pick an object and find if for the same values of attributes, it leads to some other decision class or not. We find that objects X5, X6 and X10 always lead to same decision class i.e. the concept under consideration for attribute "b". On the other hand objects X2 and X3 do not lead to Concept X, so they will not be part of lower approximation. Similarly some occurrences of objects X1, X4, X7, X8 and X9 also lead to different decision class than X, so they

also be excluded from lower approximation. So, we can with surety say that objects X5, X6 and X10 belong to the lower approximation.

As discussed earlier, semantically both definitions are same but computationally the proposed definition is more effective as it avoids construction of equivalence class structure. It directly calculates the objects belonging to lower approximation by checking objects that always lead to same decision class under consideration. Directly calculating the lower approximation in this way lets us exclude complex equivalence class structure calculation which makes algorithms using this measure more efficient. Using conventional definition on the other hand requires three steps discussed in previous section. Performing these steps offers a significant performance bottleneck to algorithms using this measure.

5.2.2 Redefined Upper Approximation:

Upper approximation defines the set of objects that can only be classified as possible members of X w.r.t. the information contained in attribute(s) $c \in C$. For attribute(s) $c \in C$ and concept X, the lower approximation will be:

$$\bar{C}X = \{X | [X]_B \cap X \neq 0\}$$

i.e. upper approximation is the set of objects that may belong to the concept X w.r.t. information contained in attribute(s) c.

On the bases of the same concept a new definition of upper approximation was proposed as follows:

$$\overline{C}X = \underline{C}X \cup \{\forall x \in U, c \in C,\ a \neq b |\ x_{\{c\}} \rightarrow a,\ x'_{\{c\}} \rightarrow b\} \quad (5.3)$$

This definition will be read as follows:

Provided that that objects x and x' are indiscernible wr.t. to attribute(s) c, they will be part of an upper approximation if either they belong to lower approximation or at least one of their occurrences leads to decision class belonging to concept X. So objects x and x' belong to upper approximation if both occurrences of them lead to different decision class for the same value of attributes. E.g. in Table-5.10 the objects X1, X4, X7,X8 and X9 lead to different decision class for

same value of attribute “b”. So all of them can be possible members of upper approximation of concept Z=2.

As with redefined lower approximation, the proposed definition for the upper approximation is semantically the same as conventional upper approximation. However, it helps us directly calculate objects belonging to upper approximation without calculating the underlying equivalence class structures.

Like the conventional definition of lower approximation, calculating upper approximation requires three steps again. We will again use Table-1 as sample to calculate upper approximation using both definitions. We suppose the concept: $X = \{x \mid Z(x) = 2\}$ as shown in example of lower approximation.

Step-1: Calculate the objects belonging to the concept X. Here is our case concept $X = \{x \mid Z(x) = 2\}$

We have already performed this step and calculated the objects belonging the concept X on the basis of decision class in previous example. So,

$\mathrm{X} = \{X4, X5, X6, X10\}$

Step-2: Calculating equivalence classes using conditional attributes.

This step has also been calculated before and objects belonging to $[X]_c$ were identified as follows by considering attribute “b”

$[X]_c = \{X1, X4, X7, X8, X9\}\{X2, X3\}\{X5, X6, X10\}$

Step-3: find the objects belonging to upper approximation

In this step we finally calculate the objects belonging to lower approximation. The step is calculated by identifying the objects in $[X]_c$ that have non-empty interaction with objects belonging to concept X. the intention is to identify all those objects at least one instance of which leads to the decision class belonging to the concept X. Mathematically $[X]_c \cap X \neq 0$. In our case:

$\bar{C}X = \{X1, X4, X7, X8, X9\}\{X5, X6, X10\}$

Proposed definition of upper approximation avoids calculating computational expensive step of indiscernibility relation. It simply scans the entire dataset for the concept X using the attributes c and finds all the indiscernible objects such that any of their occurrence belongs to the concept X. At least one occurrence should lead to decision class belonging to concept X. In our case, objects X1, X4, X7, X8 and X9 are indiscernible and at least one of their occurrence leads to concept Z=2. Objects X2 and X3 are indiscernible but none of their occurrence leads to required concept, so they will not be part of upper approximation. Objects X5, X6 and X10 belong to lower approximation so they will be part of upper approximation as well. In this way using the proposed definition, we find the following objects as part of upper approximation:

$\bar{C}X = \{X1, X4, X7, X8, X9\}\{X5, X6, X10\}$

That is the same as produced by using the conventional method.

Semantically proposed definition of upper approximation is same as conventional one i.e. it produces the same objects that may possibly be classified as members of concept X. However, it calculates these objects without calculating equivalence classes. So, the proposed approach is computationally less expensive which consequently can result in enhanced performance of the algorithms using proposed method. The conventional definition on the other hand again requires three steps (as discussed above) to calculate upper approximation which impose computational implications.

5.3 Redefined Preliminaries based Feature Selection (RPFS)

Based on our redefined approximation preliminaries, we have proposed a new feature selection algorithm. The proposed algorithm uses lower approximation based dependency measure to calculate dependency. Dependency defines how uniquely the value of attribute "D" is determined by value of "C". Conventional positive region based dependency measure uses indiscernibility based equivalence class structures to calculate lower approximation for all decision classes. In our proposed feature selection algorithm, instead of calculating indiscernibility relation for lower approximation, we have used our proposed definition. Calculating dependency in this way results in significant increase in performance of the algorithm based on this approach. Our proposed algorithm has two main features:

1. Instead of using conventional dependency measure, it calculates dependency by using the proposed lower approximation method.

2. It selects attributes on the basis of decreasing order of dependency i.e. attribute with highest degree of dependency is selected first, and then attribute with second highest degree of dependency and so on.

The proposed redefined preliminaries based feature selection (RPBFS) algorithm is a two-step process:

1. Calculate the dependency of decision attribute on each individual conditional attribute $c \in C$.

2. Select the next potential candidate attribute having highest degree of dependency.

Figure-5.2 shows the diagrammatic representation of the proposed solution.

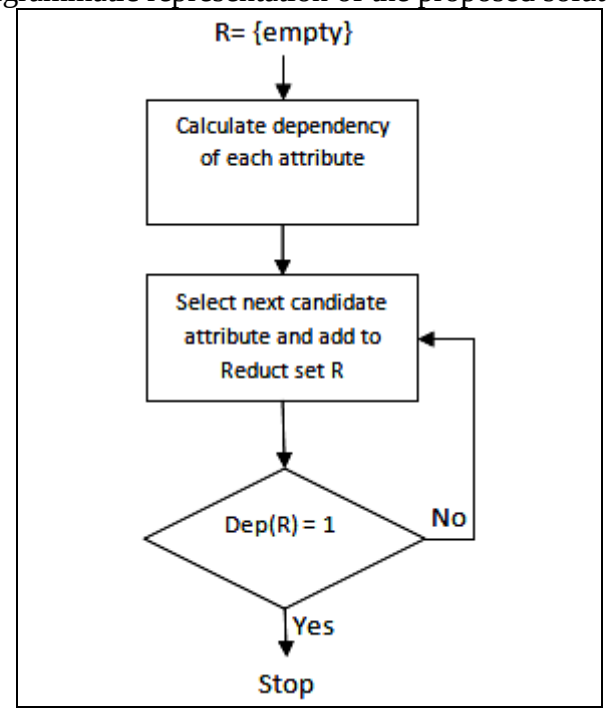


Figure-5.2: Algorithmic flow of proposed Feature selection approach

Algorithm initially starts with empty Reduct set and calculates dependency of each individual conditional attribute $c \in C$. Then the attribute having next maximum dependency value is selected and added to reduct set R. Now the dependency of R is calculated using proposed lower approximation based method. If dependency of R is equal to one (1) or $\gamma(R, D) = \gamma(C, D)$, algorithm stops by outputting R as required Reduct set. Figure-5.3 shows the pseudo code of the algorithm.

C: C1,C2,…..Cn set of conditional attributes
D: Decision attribute

(a) R ←{}
(b) $\forall c \in C$
$calculate\ \gamma_i$ where i=1,2,3….n
(d) Do
(e) Select max(γ_i) //where ith attribute is the one with maximum dependency
(f) R ← R ∪ $\{X_i\}$
(g) While $\gamma(R, D) <> \gamma(C, D)$
(h) Return R

Figure-5.3: Proposed algorithm

The proposed algorithm is based on the following two assumptions:

1. If we combine an attribute of higher dependencies with another attribute of higher dependency then it is more likely that the dependency of resulting set will be more than if we combine attribute of higher dependency with the attribute of lower dependency. For example, consider Table-5.9 given below:

Table-5.9: Sample decision system

U	a	b	c	d	Z
X1	L	3	M	H	1
X2	M	1	H	M	1
X3	M	1	M	M	1
X4	H	3	M	M	2
X5	M	2	M	H	2
X6	L	2	H	L	2
X7	L	3	L	H	3
X8	L	3	L	L	3
X9	M	3	L	M	3
X10	L	2	H	H	2

Here:

$\gamma(\mathrm{B}, \mathrm{Z}) = 0.5$

$\gamma(\mathrm{C}, \mathrm{Z}) = 0.3$

$\gamma(\mathrm{A}, \mathrm{Z}) = 0.1$

$\gamma(D, Z) = 0.0$

Now if we combine attribute "B" with "C" the dependency is likely to be more than if we combine "B" with "A". Similarly, combining "B" with "A" will likely result more increase in dependency than if we combine "B" with "D".

So using the above IS:

$\gamma(\{B \cup D\}, Z) = 0.6$

$\gamma(\{B \cup C\}, Z) = 0.8$

$\gamma(\{B \cup A\}, Z) = 0.7$

2. We have observed that dependency of union of two attributes is always greater than or equal to the maximum dependency of any of these attributes. However, there is very rare chance that dependency of union of two attributes being equal to that of the maximum one. Most of the time dependency of resultant subset increases.

Example:

Proposed solution is explained with help of an example in this section We will consider the decision system given in table-5.12.

Algorithm starts initially with empty Reduct set and computes dependency of each individual attribute which in our case will be as follows:

$R \leftarrow \{\}$

$\gamma(A, Z) = 0.1$

$\gamma(B, Z) = 0.5$

$\gamma(C, Z) = 0.3$

$\gamma(D, Z) = 0.0$

Now we will select the attribute with maximum dependency and will be made part of Reduct set. Here in our case it is Attribute "B", so Reduct set becomes:

R={B}

We will then see if dependency of Reduct set is equal to that of entire dataset. So far it is not, so we will select next attribute with maximum dependency, it is "C", so R becomes:

R={B,C}

Again dependency of R is evaluated. So far condition is false so we will select next attribute with highest dependency value which is "A". Reduct set now becomes:

R = {B,C,A}

Now the condition becomes false, and algorithm will output R = {B,C,A} as required Reduct set.

The proposed method provides a lightweight feature selection method which attempts to find feature subset based on the value of attribute dependency which makes it effective in three ways:

1. Selecting only the attributes with higher dependency makes it possible to avoid exhaustive search and find feature subset with minimum effort which makes the approach computationally efficient.

2. Efficiency of algorithm further increases by calculating dependency with proposed preliminaries calculation method.

3. Selecting attributes based on value of attribute dependency ensures that resulted feature subset is optimal and there is no irrelevant or redundant attribute.

5.4 Summary

Conventional rough set based dependency measure requires three steps to calculate dependency of a decision attribute "D" on conditional attribute(s) "C". The process consists of calculating equivalence class structure using decision attribute, equivalence class structure using conditional attribute and finally positive region calculation. The overall process is computationally too expensive to make the positive region based approaches inappropriate for large datasets. To overcome the problem, we have proposed two alternate methods for calculating rough set based dependency measure called Incremental Dependency Classes (IDC) and Direct Dependency Classes (DDC). Both of the approaches use different rules to calculate dependency as we read each new record in dataset. We have discussed in details with examples about how to calculate dependency using both of these approaches. In next section we will discuss feature selection based on IDC and DDC.

Chapter 6: Feature Selection using Dependency Classes

We have integrated dependency classes with various feature selection algorithms which in their original form were using positive region based rough set dependency measure. We have replaced all the steps using positive region based dependency measure with dependency classes. In this section we will explain each of these algorithms.

6.1 Feature Selection Using Incremental Dependency Classes

Incremental Dependency Classes (IDC) can be used in any feature selection algorithm, by simply replacing the positive region based dependency calculation with IDC. Here we have selected seven most popular feature selection algorithms which are re-implemented with IDC. All the steps where positive region based roughest dependency measure was used, were replaced with IDC based dependency measure.

6.1.1 Genetic Algorithm Using IDC

Genetic Algorithm using IDC or in short GA (IDC) has all the features of conventional genetic algorithms like crossover, mutation and fitness evaluation function etc. as given by [24]. However, the slight changes made here were that IDC based dependency measure was used instead of positive region based approach and crossover order was based on decreasing order of chromosome dependency. The chromosome that shows the ideal fitness score i.e. dependency=1, at any generation, was considered to be the optimum solution.
The main features of the proposed GA (IDC) are as follows:

- Each chromosome represents a subset of candidate features that can possibly be a Reduct set.
- Fitness function was based on positive region based dependency.
- The chromosomes were selected for crossover in decreasing order of dependency as shown in Table-6.1.

Table- 6.1: Example of Chromosomes crossover order in GA(IDC)

Chromosome No.	Chromosome	Fitness score	Crossover order
1	a,b,c,d	0.86	Chromosome no. 1 will mate with Chromosome no. 3
2	l,m,n,o	0.45	-
3	s,t,u,v	0.72	-
4	w,x,y,z	0.61	Chromosome no. 4 will mate with Chromosome no. 2

The fitness score of each chromosome represents the dependency of the decision attribute. Cross over with decreasing order of dependency makes results in quality offspring chromosomes having higher fitness as compared to their parents. This is based on our observation that combining the attributes with higher degree of dependency result in rapid increase in dependency of resultant attributes as compared to combining those having low degree of dependency. So with above mentioned crossover order higher quality chromosomes are generated in lesser number of iterations.

Now we explain the execution of algorithm with example using "hepatitis" dataset taken from UCI repository [26]. Initial chromosome size was equal to the number of attributes in dataset, a gene in each chromosome used to represent presence of attribute. Gene were represented using the sequence number of the attribute in dataset whereas absence of attribute was represented by "-1". Initial population size was set to "10" i.e. at any stage ten chromosomes were used. User can specify any population size of 2*n where n=1, 2, 3…. N. Initially the selected population is shown in the Figure-6.1. As GA(IDC) performs crossover in descending order of fitness value (which actually represents the dependency value of the decision attribute on conditional attributes represented by genes of the chromosome). So, chromosome no. 5 and 9 (Figure-6.2) having highest dependency values among all population, will cross over with each other.

	Chromsome	Fitness
1	-1,-1,-1,-1,-1,7,-1,-1,-1,-1,-1,-1,-1,15,-1,-1,-1,-1,-1,	0.025
2	-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1	0
3	-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1	0
4	-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,14,-1,-1,-1,-1,-1,-1	0
5	-1,-1,-1,-1,-1,-1,-1,-1,-1,11,-1,-1,-1,-1,-1,17,-1,-1,-1	0.8375
6	-1,-1,4,-1,-1,-1,8,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1	0.15
7	-1,-1,-1,5,-1,-1,-1,-1,-1,-1,-1,-1,14,-1,-1,-1,-1,-1,-1	0
8	-1,-1,4,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1	0
9	-1,-1,-1,5,-1,-1,8,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,20	0.475
10	-1,-1,-1,-1,-1,7,-1,9,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1	0.1625

Figure-6.1: Initial Population.

	Chromsome	Fitness
1	-1,-1,-1,-1,-1,7,-1,-1,-1,-1,-1,-1,-1,15,-1,-1,-1,-1,-1,	0.025
2	-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1	0
3	-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1	0
4	-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,14,-1,-1,-1,-1,-1,-1	0
5	-1,-1,-1,-1,-1,-1,-1,-1,-1,11,-1,-1,-1,-1,-1,17,-1,-1,-1	0.8375
6	-1,-1,4,-1,-1,-1,8,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1	0.15
7	-1,-1,-1,5,-1,-1,-1,-1,-1,-1,-1,-1,14,-1,-1,-1,-1,-1,-1	0
8	-1,-1,4,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1	0
9	-1,-1,-1,5,-1,-1,8,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,20	0.475
10	-1,-1,-1,-1,-1,7,-1,9,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1	0.1625

Figure-6.2: chromosome with highest dependency

The chromosome with same colour will crossover with each other. Chromosome no. 6 and 10 are next to crossover and so on. The resulted Offspring population is shown in Figure-6.3 below:

1	-1,-1,-1,-1,-1,-1,-1,-1,-1,11,-1,-1,-1,-1,-1,-1,-1,20
2	-1,-1,-1,5,-1,-1,8,-1,-1,-1,-1,-1,-1,-1,-1,17,-1,-1,-1
3	-1,-1,-1,-1,-1,7,-1,9,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
4	-1,-1,4,-1,-1,-1,8,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
5	-1,-1,-1,-1,-1,7,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
6	-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,15,-1,-1,-1,-1,-1
7	-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,14,-1,-1,-1,-1,-1,-1
8	-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
9	-1,-1,-1,5,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
10	-1,-1,4,-1,-1,-1,-1,-1,-1,-1,-1,-1,14,-1,-1,-1,-1,-1,-1

Figure-6.3: Offspring after first crossover

	Chromsome	Fitness
1	-1,-1,-1,-1,-1,-1,-1,-1,-1,11,-1,-1,-1,-1,12,-1,-1,-1,20	0.15
2	-1,-1,-1,5,-1,-1,8,-1,-1,-1,-1,-1,-1,3,-1,17,-1,-1,-1	0.925
3	-1,-1,-1,-1,-1,7,-1,9,-1,-1,-1,-1,15,-1,-1,-1,-1,-1,-1	0.1875
4	-1,-1,4,-1,-1,15,8,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1	0.2
5	-1,-1,-1,-1,-1,7,-1,-1,-1,-1,-1,-1,-1,-1,20,-1,-1,-1	0.4
6	-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,15,8,,1,-1,-1,-1	0.1875
7	-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,12,-1,14,-1,-1,-1,-1,-1,-1	0.025
8	-1,-1,-1,-1,-1,-1,-1,-1,-1,13,-1,-1,-1,-1,-1,-1,-1,-1,-1	0
9	-1,8,-1,5,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1	0.15
10	-1,-1,4,-1,-1,-1,-1,-1,-1,-1,-1,-1,14,-1,-1,-1,-1,5,-1	0.125

Figure-6.4: Offspring after mutation

The red highlighted genes show the positions where mutation took place in Figure-6.4. To keep things simple, uniform mutation operator was used in which a random gene is selected and replaced with a random value that represents a valid attribute. Process continues until we find a chromosome with highest fitness value. Algorithm checks in each population, if any of the chromosomes has fitness value of "1". Algorithm stops if any of such chromosome is found. However if after "α"

number of generations, algorithm fails to find such chromosome, it stops with highest fitness value chromosome as the output. In case of our example algorithm stopped after fourth generation. Figure-6.5 shows the population in fourth generation along with the highlighted best chromosome:

	Chromsome	Fitness
1	-1,-1,-1,13,-1,7,-1,-1,-1,-1,-1,-1,-1,15,8,-1,-1,-1,-1	0.7
2	-1,-1,11,-1,-1,-1,-1,-1,-1,19,-1,14,-1,3,-1,17,-1,-1,-1	1
3	-1,-1,13,-1,-1,-1,-1,-1,-1,-1,-1,-1,15,-1,-1,-1,-1,-1,-1	0.075
4	-1,-1,4,-1,-1,15,8,-1,-1,20,-1,-1,-1,-1,-1,16,-1,10,-1	0.975
5	-1,-1,-1,5,11,-1,8,-1,-1,-1,12,-1,14,-1,-1,-1,-1,4,-1	0.5875
6	9,-1,4,-1,-1,-1,-1,-1,-1,13,-1,-1,-1,-1,-1,20,-1,-1,-1	0.4875
7	-1,11,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,14,-1,-1,-1,-1,5,-1	0.0125
8	-1,8,-1,9,-1,-1,-1,-1,12,-1,-1,-1,-1,-1,12,10,-1,-1,20	0.5875
9	-1,-1,-1,-1,4,7,-1,9,-1,10,-1,-1,-1,-1,-1,-1,-1,-1,-1	0.325
10	-1,-1,-1,-1,-1,-1,-1,14,-1,-1,-1,-1,20,-1,-1,-1,-1,-1,-1	0.0125

Figure-6.5: Last generation.

Chromosome no. 2 is best one, so our reduct set will be:

R = {3, 11, 14, 17, 19}

Figure-6.6 shows the best chromosome in each generation along with its fitness value. Note that how fitness increased with each generation:

Chrosome	Fitness	Gen.
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,11,-1,-1,-1,-1,-1,17,-1,-1,-1,	0.8375	1
-1,-1,-1,5,-1,-1,8,-1,-1,-1,-1,-1,-1,3,-1,17,-1,-1,-1,	0.925	2
-1,-1,-1,-1,-1,-1,7,-1,-1,-1,-1,-1,14,-1,3,-1,17,-1,-1,-1,	0.975	3
-1,-1,11,-1,-1,-1,-1,-1,-1,19,-1,14,-1,3,-1,17,-1,-1,-1,	1	4
End here (optimum)		

Figure-6.6: Best chromosomes in all generations

Thus, GA(IDC) results in crossover of higher dependency value chromosomes with each other, which means that required feature set is likely to be obtained with fewer generations.

6.1.2 QuickReduct Algorithm Using IDC

QuickReduct algorithm using IDC, QR(IDC) is modified form of conventional QuickReduct (QR) algorithm [2]. In QR (IDC) the step of calculating positive region based rough set dependency was replaced with IDC based dependency measure. Rest of the algorithm was same. Figure 6.7 highlights the points where the conventional dependency calculation step was replaced with incremental dependency calculation method.

QUICKREDUCT (C,D)
C, The set of all conditional features
D, The set of decision features
(10) $R \leftarrow \{\}$
(11) do
(12) $T \leftarrow R$
(13) $\forall x \in (C - R)$
(14) If $\gamma_{R\cup\{x\}}(D) > \gamma_T(D)$
(15) $T \leftarrow R \cup \{x\}$
(16) $R \leftarrow T$
(17) until $\gamma_R(D) == \gamma_C(D)$
(18) Output R

Figure-6.7:Quickreduct algorithm taken from [14]

6.1.3 ReverseReduct Algorithm Using IDC

ReverseReduct [2] approach is not often used for large datasets, as the algorithm must evaluate large feature subsets (starting with the set containing all features), which is too costly, although the computational complexity is, in theory, the same as that of forward-looking QuickReduct. In ReverseReduct algorithm using IDC ReverseReduct (IDC), step of calculating the positive region based dependency calculation step was replaced with IDC based method.

6.1.4 Incremental Feature Selection Algorithm (IFSA) using IDC

IFSA [25] is intended for feature selection in datasets where features vary dynamically in decision systems. In its original version, IFSA uses conventional dependency measure where dependency is calculated using positive region. Figure-6.8 shows the IFSA with highlighted steps where IDC were used instead of positive region based approach.

Input A decision system $DS = (U, A = C \cup D)$, the original feature subset Red, the original positive region $\gamma_C(D)$, and the adding feature set C_{ad} or the deleting feature set C_{de}, where $C_{ad} \cap C = \varnothing$, $C_{de} \subseteq C$;
Output A new feature subset Red'.
Begin
1) Initialize $P \leftarrow Red$;
2) If a feature set C_{ad} is added into the system DS;
3) let $C' \leftarrow C \cup C_{ad}$;
4) compute the equivalence classes U/C' and $\gamma_{C'}(D)$;
//According to Theorem 1
5) for $i = 1$ to $|C_{ad}|$ do
6) compute $sig_1(c_i, C_{ad}, D)$;
7) if $sig_1(c_i, C_{ad}, D) > 0$, then $P \leftarrow P \cup \{c_i\}$;
8) end for
9) if $\gamma_P(D) = \gamma_{C'}(D)$, turn to Step 25; else turn to Step 16;
10) End if
11) If an feature set C_{de} are deleted from the system DS;
12) let $C' \leftarrow C - C_{de}$;
13) if $C_{de} \cap P = \varnothing$, turn to Step 25; else $P \leftarrow P - C_{de}$ and turn to Step 14;
14) compute the equivalence classes U/C' and $\gamma_{C'}(D)$;
//According to Theorem 2
15) End if
16) For $\forall c \in C' - P$, construct a descending sequence by $sig_2(c, P, D)$, and record the result by $\{c'_1, c'_2, \ldots, c'_{|C'-P|}\}$;
17) While $(\gamma_P(D) \neq \gamma_{C'}(D))$ do
18) for $j = 1$ to $|C' - P|$ do
19) select $P \leftarrow P \cup \{c'_j\}$ and compute $\gamma_P(D)$;
20) End while
21) For each $c_j \in P$ do
22) compute $sig_1(c_j, P, D)$;
23) if $sig_1(c_j, P, D) = 0$, then $P \leftarrow P - \{c_j\}$;
24) End for
25) $Red' \leftarrow P$, return Red';
End

Figure-6.8: IFSA taken from [25]

6.1.5 Supervised PSO Based Quick Reduct (PSO-QR) Using IDC

Supervised PSO based QuickReduct [23] is a supervised hybrid feature selection algorithm based on Particle Swarm Optimization (PSO) and rough sets. In its original form, it uses positive region based dependency measure for calculating the fitness of the particles. Algorithm suffers the same drawbacks as by the others due to calculation of positive region. In PSO-QR (IDC) we have calculated dependency using IDC. Figure-6.9 highlights the pints where positive region based dependency measure was replaced with IDC based dependency.

Input: C, the set of all conditional features,
D, the set of decision features.
Output: Reduct R

Step 1: Initialize X with random position V_i with random velocity
$\forall: X_i \leftarrow randomPosition();$
$V_i \leftarrow randomVelocity();$
Fit ← 0; globalbest ← Fit;
Gbest ← X_1; Pbest(1) ← X_1
For i = 1 … S
pbest(i) = X_i
Fitness(i) = 0
End For

Step 2: While Fit != 1 // stopping criterion
For i = 1 … S // for each particle
$\forall: X_i;$
//Compute fitness of feature subset of X_i
R ← Feature subset of X_i (1's of X_i)
$\forall\, x \in (C - R)$
$\gamma_{RU(X)}(D) = \frac{\left|POS_{RU(X)}(D)\right|}{|U|}$
$Fit = \gamma_{RU(X)}(D)\ \forall\, xcR, \gamma_x(D) \neq \gamma_c(D)$
End For

Step 3: Compute best fitness
For i = 1:S
if(Fitness(i) > globalbest) // if current fitness is greater than global best fitness
globalbest ← Fitness(i); // assign current fitness value as global best fitness
gbest ← X_i ;
getReduct(X_i)
Exit
End if

End for
UpdateVelocity(); // Update velocity V_i's of X_i's
UpdatePosition(); // Update position of X_i's
// Continue with the next iteration
End {while}
Output Reduct R

Figure-6.9: PSO-QR taken from [23]

6.1.6 Fish Swarm Algorithm (FSA) using IDC

FSA [26] uses the idea of fish swarm for rough set reduction problem. Like other algorithms, in its original form, it uses positive region based dependency measure as part of fitness function. However, in Fish Swarm Algorithm (FSA) using IDC i.e. FSA (IDC) we replaced this step with IDC based dependency calculation method. Figure-6.10, 6.11, 6.12 and 6.13 highlight the steps where positive region based dependency was replaced with IDC based measure.

Input: a decision system $\mathcal{DS} = (U, C \bigcup D, V, f)$ and parameters
Output: a minimal feature reduct *Rmin* and its cardinality *Lmin*

(1) Initialize $R_{\min} = C, L_{\min} = |C|, iteration = 0$
(2) Compute $\gamma_C(D)$ by formula
(3) While($iteration <= maxcycle$)
(4) {
(5) Generate all fish
(6) For each fish $k \in Fish$
(7) {
(8) $R_k = \phi, L_k = 0$
(9) Select a feature $a_k \in C$ randomly
(10) $R_k = \{R_k \bigcup a_k\}, L_k = |R_k|$
(11) }
(12) Do
(13) {
(14) For each fish $k \in Fish$
(15) {
(16) R_s=searching(R_k)
(17) R_w=swarming(R_k)
(18) R_f=following(R_k)
(19) R_k=maxfitness(R_s, R_w, R_f)
(20) $L_k = |R_k|$
(21) If ($\gamma_{R_k}(D) == \gamma_C(D)$, then
the fish finds a local reduct, and it exits for loop
(22) If ($\gamma_{R_k}(D) == \gamma_C(D)$ and $L_k < L_{\min}$), then
$R_{\min} = R_k, L_{\min} = L_k$
(23) }
(24) } Until each fish finds its local reduct
(25) $iteration = iteration + 1$
(26) }
(27) Output $R_{\min}$ and $L_{\min}$

Figure-6.10: FSA taken from [26]

Input: a decision system $DS = (U, C \bigcup D, V, f)$ and a current position R_k
Output: a next position R_s

(1) $i = 0, R_t = R_k, R_p = R_k$
(2) Do
(3) {
(4) Select a feature $a_k \in C - R_p$ randomly
(5) $R_t = \{R_k \bigcup a_k\}$
(6) $i = i + 1, R_p = \{R_p \bigcup a_k\}$
(7) } Until fitness(R_t) > fitness(R_k) or $i >= |C - R_k|$
(8) $R_s = R_t$
(9) Return the next position R_s

Figure-6.11: FSA Searching algorithm taken from [26]

Input: a decision system $DS = (U, C \bigcup D, V, f)$ and a current position R_k
Output: a next position R_w

(1) $n = 0, R_t = \phi, v = 0$
(2) For each friend fish R_i
(3) {
(4) If distance(R_k, R_i) < *visual*, then
$n = n + 1, R_t = R_t + R_i, v = v +$ fitness(R_i)
(5) }
(6) R_c=center(R_t)
(7) If $v/n < \delta *$ fitness(R_k), then $R_w = R_c$ else R_w=searching(R_k)
(8) Return the next position R_w

Figure-6.12: FSA Swarming algorithm taken from [26]

Input: a decision system $DS = (U, C \bigcup D, V, f)$ and a current position R_k
Output: a next position R_f

(1) $n = 0, R_{max} = \phi, v_{max} = 0$
(2) For each friend fish R_i
(3) {
(4) If distance(R_k, R_i) < *visual* and fitness(R_i) > fitness(R_{max}), then
$R_{max} = R_i$
(5) }
(6) For each friend fish R_i
(7) {
(8) If distance(R_{max}, R_i) < *visual*, then
$n = n + 1, v_{max} = v_{max} +$ fitness(R_i)
(9) }
(10) If $v_{max}/n < \delta *$ fitness(R_k), then
$R_f = R_{max}$ else R_f=searching(R_k)
(11) Return the next position R_f

Figure-6.13: FSA fitness algorithm taken from [26]

6.1.7 Rough Set Improved Harmony Search Quick Reduct Using IDC

Rough Set Improved Harmony Search Quick Reduct (RS-IHS-QR) [27] is a rough set based hybrid algorithm that combines QuickReduct with improved harmony search method for feature selection. The algorithm uses rough set based dependency measures as its objective function. In Harmony Search Quick Reduct using IDC i.e. RS-IHS-QR (IDC), we replaced the objective function with rough set based dependency measure using IDC i.e. Dep (IDC). Figure-6.14 shows the steps where positive region based dependency was replaced with IDC based dependency.

6.2 Parameter Settings

QuickReduct and ReveseReduct are exhaustive algorithms having no special parameters. For the rest of algorithms, the parameter details, their values and effect on algorithms is discussed in following section.

6.2.1 Particle Swam Optimization Based QuickReduct Using IDC PSO-QR(IDC):

In PSO-QR(IDC), inertia weights "w" were set between range 0.9 to 1.2 as given by [34] because with this range of "w" there are lesser chances of algorithm to fail to find the global optimum within a reasonable number of iterations. Range $\left[1, \frac{1}{N} \times Number\ of\ Attribute\right]$ was used for velocity because the particles with velocity above this range fly far from optimal solution.

Input : C, the set of conditional attributes; D, the set of decision attributes
Output : Best Reduct (feature subset)
Step 1: Define the fitness function, f(X) // as in Eq. (12)
Initialize the variables HM=10 // Harmony Memory (Population)
HMCR = 0.95 // Harmony Memory Consideration Rate (For improvisation)
NI = 100 // Maximum number of Iterations,
PVB //Possible value bound of X
PAR_{min} = 0.4; PAR_{max} = 0.9;
bw_{min} = 0.001; bw_{max} = 0.1; // Pitch Adjusting Rate & bandwidth ∈(0 to 1)
fit =0;X_{old}= X_1; bestfit =X_1;bestreduct={};
Step 2: Initialize Harmony Memory, HM = (X_1,X_2 ....X_{HMS})
For i = 1 to HM //for each harmony
∀ : X_i ; // X_i is the i^{th} harmony vector of HM
T ← { };
//Compute fitness of feature subset of X_i
R← Feature subset of X_i (1's of X_i)
∀x ∈ R
$\gamma_{T\cup\{x\}}(D) = \frac{|POS_{T\cup\{x\}}(D)|}{|U|}$
$f(X_i) = \gamma_{T\cup\{x\}}(D) \quad \forall X \subset R, \gamma_X(D) \neq \gamma_C(D)$
if $f(X_i)$ > fit
fit ← $f(X_i)$
X_{old} ← X_i
End if
End for
Step 3: Improvise new Harmony Memory
While iter ≤ NI or fit ==1// Stopping Criterion
for j=1,2,...NVAR
∀:X_{old}(j) // x is the variable of X
Update Pitch Adjusting Rate();
Update bandwidth();
if rand ( ) ≤ HMCR // rand ∈ [0,1]
// Construct the new harmony Xnew from the best harmony vector Xold.
X_{new} ← X_{old}; // assigning the best harmony to the new harmony
if rand ( ) ≤ PAR // rand ∈ [0,1]
X_{new}(j) = X_{new}(j) ± rand() * bw
end if
else
//Choose a random value of variable Xnew
$X_{new}(j)=PVB_{lower}$ + rand( ) * (PVB_{upper} – PVB_{lower})
end if
end for
Step 4: Update the new Harmony Memory
// Compute fitness function for New Harmony X_{new} as defined in Step 2.
if $f(X_{new}) \geq f(X_{old})$
X_{old}← X_{new};// Accept and replace the old harmony vector with new harmony.
if $f(X_{new})$ > fit
fit ← $f(X_{new})$;//Replacing Worstfit with bestfit
bestfit ← X_{new};
End if
Exit
end if
//Continue with the next iteration
end while
bestreduct ← feature subset of bestfit
// Reduced feature subset: 1's of bestfit

Figure-6.14: RS-IHS-QR taken from [27]

6.2.2 Genetic Algorithm Using IDC GA(IDC)

In GA(IDC) chromosome size was set to total number of found by QuickReduct algorithm to ensure unbiased analysis. However, any encoding scheme can be used here. For mutation, one point uniform mutation scheme was used in which a random gene is replaced with another (randomly selected) gene representing an attribute. The reason behind was that in each generation, decreasing order of dependency already resulted in high quality off-springs, so one point mutation was considered to be sufficient in this regards.

6.2.3 Fish Swarm Algorithm (FSA) Using IDC

For FSA (IDC), all the parameters were used with their original value. However, to ensure unbiased analysis, both FSA and FSA (IDC) were initialized with same fish positions. Furthermore, stopping criteria was also slightly updated. The algorithm was made to terminate as soon as the first fish found its optimal positions (i.e. fitness of "1"). This step was taken to complete the algorithm as soon as possible for large datasets.

6.2.4 Rough Set Improved Harmony Search Quick Reduct using IDC:

Just like FSA and FSA (IDC), both RS-HIS-QR and RS-HIS-QR (IDC) were initialized with same harmony memory to avoid biasedness in comparison. No change was made in rest of the parameters.

6.3 Feature Selection Using DDC

DDC can also be used in any feature selection algorithm, by simply replacing the positive region based dependency calculation with DDC. Just like in IDC, we have re-implemented all of the various algorithms discussed in related work section using DDC approach. Here we will discuss these algorithms in short, as in previous sections these have already been discussed in detail.

6.3.1 Supervised PSO Based Quick Reduct Using DDC:

Supervised PSO based Quick Reduct [23], PSO-QR was originally designed to use positive region based dependency measure. We re-implemented algorithm using direct dependency calculation method.

6.3.2 Genetic Algorithm Using DDC:

Genetic Algorithm using DDC, GA(DDC) is the same as mentioned in section 6.1.1. The only change made was that DDC based methods was used in fitness function in contrast with original positive regional based dependency calculation method.

6.3.3 Incremental Feature Selection Algorithm (IFSA) Using DDC:

IFSA [23], in its original form, uses positive region based dependency measure. Just like IDC based IFSA, we re-implemented it with DDC based method. All the steps calculating positive region based dependency were replaced with DDC based method. Rest of the details of the algorithm were kept intact.

6.3.4 Fish Swarm Algorithm (FSA) Using DDC:

FSA [24] used swarm based optimization to perform feature selection. Algorithm used positive region based dependency measure for all searching, swarming and following behaviour. Stopping criteria was also based on positive region based approach. We replaced all the positive region based steps with DDC.

6.3.5 Rough Set Improved Harmony Search Quick Reduct (RS-IHS-QR) Using DDC

RS-IHS-QR [25] proposes a hybrid approach for feature selection based on Rough set theory combined with improved harmony search algorithm. Just like other positive region based methods, it also uses conventional positive region based dependency measure. We, however, made this algorithm to work with DDC based method.

6.4 Parameter settings

Parameter settings were for DDC was kept same as that in case of IDC. No further change was made apart from those mentioned in section 6.3. Similarly stopping criteria was also kept same.

6.5 Summary

In this section we have discussed various algorithm that were used with IDC and DDC. Originally these algorithms were designed for positive region based rough set dependency measure. However, we re-implemented these algorithms to work with proposed dependency calculation methods. In next section we will discuss results and analysis of the proposed dependency calculation methods and feature selection techniques based on these methods.

Chapter 7: Results and Analysis

A comparison framework consisting of three components, percentage decrease in execution time to generate the output, memory used and percentage accuracy is used to justify the validity and effectiveness of both IDC and DDC. Experiments are conducted using various publically available datasets from the UCI repository.

7.1 Comparison Framework

The experiments to justify the validity and effectiveness of both IDC and DDC were conducted in two steps. In first step dependency calculated using both IDC and DDC itself were verified as per components of comparison framework. After this, comparison was carried out between the algorithms using IDC and DDC and those using positive region. Here we discuss all these components one by one.

7.1.1 Percentage Decrease in Execution Time

Percentage decrease in execution time specifies the ecy of an algorithm in terms of how fast it is and how much execution time it cuts down. For this purpose system stop watch was used, which after feeding the input was started and after getting the results was stopped. The formula to calculate the % decrease is as follows:

$$Percentage\ Decrease = 100 - \frac{E(1)}{E(2)} * 100 \quad (7.1)$$

Where E(1) is execution time of one algorithm and E(2) is that of its competitor. However, as GA, PSO-QR, FSA and RS-IHS-QR can have different number of iterations, in different executions, based on the fitness of results, so, for these algorithms, the average execution time of single iteration was considered while keeping the other parameters as same in both cases i.e. using positive region and using IDC.

7.1.2 Memory Usage

Memory usage specifies the maximum amount of runtime memory taken by the algorithm to complete the task taken during its execution. We manually calculated the memory by summing the size of each of the intermediate data structure used.

7.1.3 Accuracy

The term accuracy implies that the IDC and DDC produce the same output as the one produced by conventional positive region based method. We compared dependency of attributes using IDC and DDC against the values generated by positive region based approach. For feature selection algorithms, we compared the Reducts found using IDC with those using positive region. However, in case GA, PSO-QR, FSA and RS-IHS-QR, accuracy was measured by analyzing the Reducts against their fitness.

7.2 Experimental Analysis: IDC

Before explaining the details of experiments performed, the details of datasets used for experimental purpose are shown in table-7.1, taken from UCI machine learning repository [26].

Table-7.1: Summary of datasets used

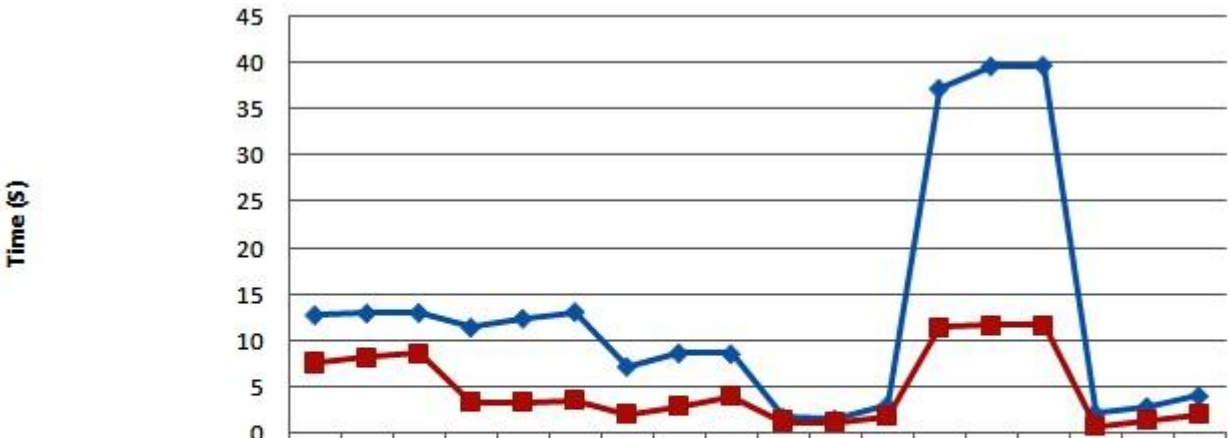
Dataset	Instances	Attributes	Decision classes	Dataset characteristics / Attribute characteristics
Gisette	6000	5000	2	Multivariate / Integer
Isolet	7797	617	26	Multivariate/ Real
Musk-2	6598	168	2	Multivariate/ Integer
UJIindoorLoc	1112	529	5	Multivariate/ Integer, Real
Egg-Eye-style	14980	15	2	Multivariate, Sequential, Time-Series / Integer, Real
Internet advertisement	3279	1558	2	Multivariate/Categorical, Integer, Real

7.2.1 Accuracy and Efficiency of IDC For Calculating Dependency

To prove the accuracy, percentage decrease in execution time and memory usage of the IDC, we calculated the dependency of different set of attributes, for example, from "Gisette" dataset, three attribute sets were taken each containing 1000, 2000 and 3000 attributes respectively. The dependency, in this step, was calculated both through conventional rough set based dependency measure using positive region method and the proposed IDC. In Table-7.2, from left to right is the dataset name and instances/number of attributes in each set on which dependency was calculated. Column three and four specify the attribute set name and number of attributes considered in that attribute set. Columns five, six, seven and eight, nine, ten specify the dependency value, execution time and memory used by positive region based approach and IDC respectively. Finally columns eleven and twelve show the percentage decrease in execution time and memory usage taken by the IDC as compared to positive region based dependency calculation method. End results clearly show the effectiveness of the proposed IDC.

Table-7.2: Conventional positive region based approach vs IDC

Dataset	Inst/ Att	Attr. Set ID	Attr. Set Size	Dep(P)			Dep(IDC)			% dec in time	% dec in memory
				Dep	Time (s)	Mem (MB)	Dep	Time (s)	Mem (MB)		
Gisette	6000/5000	G_1_1	1000	1.0	12.79	137.37	1.0	7.64	114.48	40.3%	16.7%
		AS_G_2	2000	1.0	12.99	137.37	1.0	8.22	114.48	36.7%	16.7%
		AS_G_3	3000	1	13.05	137.37	1.0	8.59	114.48	34.2%	16.7%
Isolet	7797/619	AS_T_1	200	1	11.45	231.96	1	3.330	18.35	70.9%	92.1%
		AS_T_2	400	1	12.42	231.96	1	3.340	18.35	73.1%	92.1%
		AS_T_3	600	1	13.13	231.96	1	3.620	18.35	72.4%	92.1%
Musk-2	6598/168	AS_M_1	50	1.0	7.22	166.13	1.0	2.03	4.27	71.9%	97.4%
		AS_M_2	100	1.0	8.68	166.13	1.0	2.95	4.27	66%	97.4%
		AS_M_3	150	1.0	8.6	166.13	1.0	4.04	4.27	53%	97.4%
UJIindoorLoc	1112/529	AS_U_1	300	0.998	1.75	4.72	0.998	1.29	2.25	26.3%	52.3%
		AS_U_2	400	0.998	1.55	4.72	0.998	1.17	2.25	24.5%	52.3%
		AS_U_3	500	0.998	3	4.72	0.998	1.89	2.25	37%	52.3%
Egg-Eye-style	14980/15	AS_E_1	5	1.0	37.22	856.12	1.0	11.551	0.97	69%	99.9%
		AS_E_2	9	1.0	39.63	856.12	1.0	11.65	0.97	70.6%	99.9%
		AS_E_3	13	1.0	39.71	856.12	1.0	11.68	0.97	70.6%	99.9%
Internet advertisement	3279/1558	AS_I_1	500	0.921	2.21	41.04	0.921	0.78	19.51	64.7%	52.5%
		AS_I_2	1000	0.952	2.9	41.04	0.952	1.36	19.51	53.1%	52.5%
		AS_I_3	1500	0.978	4.08	41.04	0.978	2.14	19.51	47.5%	52.5%



	G_1_1	AS_G_2	AS_G_3	AS_T_1	AS_T_2	AS_T_3	AS_M_1	AS_M_2	AS_M_3	AS_U_1	AS_U_2	AS_U_3	AS_E_1	AS_E_2	AS_E_3	AS_I_1	AS_I_2	AS_I_3
Time (s) Deps (Pos)	12.	13	13.	11.	12.	13.	7.2	8.6	8.6	1.7	1.5	3	37.	39.	39.	2.2	2.9	4.0
Time (s) Dep (Inc)	7.6	8.2	8.5	3.3	3.3	3.6	2.0	2.9	4.0	1.2	1.1	1.8	11.	11.	11.	0.7	1.3	2.1

Figure-7.1: comparison of execution time between approaches using positive region and IDC

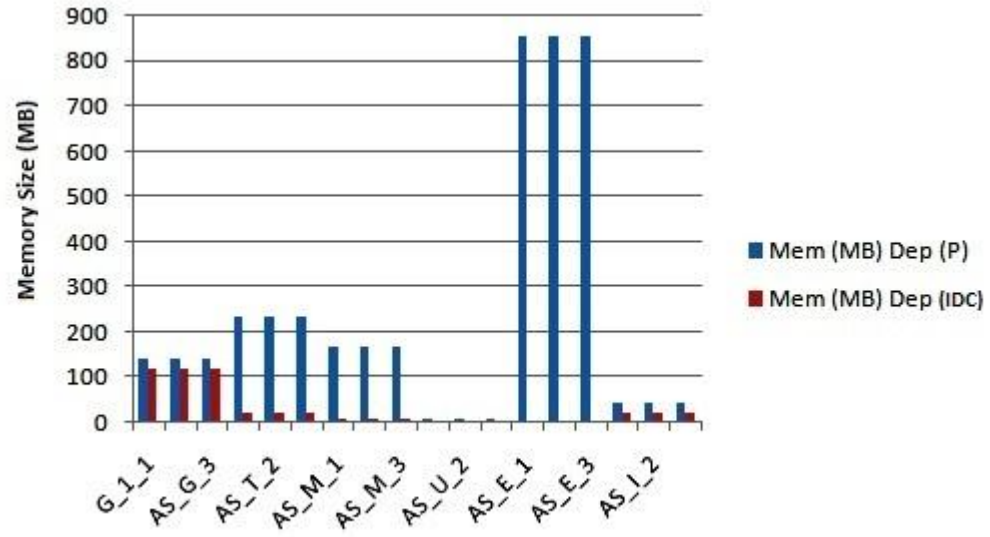


Figure-7.2: Memory comparison b/w between approaches using positive region and IDC

7.2.1.1 Percentage Decrease In Execution Time:

As it can be seen from the Table-7.2, the time consumed in case of the proposed IDC, Dep(IDC) was always less than the conventional method, Dep(p). On average, IDC reduced the execution

time by 54.5% for 18 attribute sets. The reason behind is that the IDC have successfully avoided the complex computations of calculating positive regions. We will use the decision system shown in table-5.1 to further explain this point. Calculating the dependency using positive region requires three steps. In the first step, we calculate equivalence class structure using decision attribute (Qualification in this case). In the second step, we have to calculate the equivalence class structure using condition attribute set on which dependency of decision attribute is to be calculated. It will require to match the attribute values of record i with i+1, i+2, …,i+ n. Finally we need to calculate positive region which actually calculates the cardinality of equivalence classes (based on conditional attributes) that are subsets of equivalence classes (based on decision attribute).

On the other hand, calculating the dependency using incremental dependency classes (IDC) requires only single step, in which we will match record i with i+1, i+2, …,i+n and will update the corresponding dependency variable as per the incremental dependency class the record will belong to. This leads to simpler programming logic required by Dep(IDC) as compared to Dep(P). The graph in Figure-7.1 also shows that for large datasets there is large difference in time that is required to calculate dependency, which means for large datasets, the IDC are more suitable method for calculating dependency. Based on above results and facts, we can conclude that IDC can successfully replace conventional dependency calculation method (using positive region) in any rough set based algorithm that calculate dependency of attributes “D” on “C”.

7.2.1.2 Accuracy

It can be seen from Table-7.2 that IDC show same accuracy as that of conventional approach in calculating dependency. Dependency calculated by IDC was exactly same as that of calculated by positive region based approach, while significantly cutting down the execution time.

7.2.1.3 Memory Usage

Memory taken by the IDC was less than positive region based approach in all cases. Figure-7.2 shows the memory taken by each approach in graphical form. On average almost 68.4% decrease in memory was found. To calculate the memory, we used the size of major intermediate data structures. By major we mean that the data structures that actually used to store intermediate results

and not simple variables that were used to control the logic for example loop counters, Boolean flags etc. Here we elaborate the major data structures required in both cases. We will consider the Table-5.1 to elaborate the example.

Two major matrices are required in case of positive region based approach:

1. The first one is to store the equivalence class structure of decision system w.r.t. decision attribute(s). In Table-5.1, we have two classes and maximum number of instances in any class is four (in class labeled "No"), so the required matrix will be of size 2x7 as shown below in Figure-7.3. The size of this matrix will be [number of equivalence classes x (maximum number of instances in any class+3)]. Both values can be calculated at runtime if not known in advance. First column specifies the decision class, second column specifies total number of instances in that class and third column specifies the last index having a valid value in current row. Rest of the columns contains IDs of indiscernible objects using current decision class.

Yes	3	6	X_1	X_4	X_6	-
No	4	7	X_2	X_3	X_5	X_7

Figure-7.3: Equivalence class structure w.r.t. decision attribute

2. Similarly a matrix is used to store the equivalence class structure for the current set of conditional attributes. As there are seven instances in total so we need 7x8 matrix as shown in Figure-7.4. The size of matrix will be [maximum number of instances x (total number of instances+1)]. First column stores how many instances have been added so far in current row. All other columns contain IDs of indiscernible objects using current decision class. As we cannot predict in advance that how many instances will be similar, so we have to define matrix of maximum size to cater all possible scenarios (that's why here matrix size was 7 x 8). Finally, we calculate positive region based on above mentioned matrices which checks that which equivalence classes (constructed using conditional attributes) are contained by (or are subset of) the equivalence classes constructed using decision attribute.

1	X1	-	-	-	-	-	-
1	X2	-	-	-	-	-	-
2	X3	X4	-	-	-	-	-
3	X5	X6	X7	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

Figure-7.4: Equivalence class structure w.r.t. conditional attributes

In case of IDC, on the other hand, we do not need to calculate equivalence classes (because we don't need positive region), so we simply develop a simple matrix to store instances along with their attributes to keep track of how dependency will be refreshed as shown in Figure-7.5.

S1	Doctorate	Yes	1	1
S1	Diploma	No	1	1
S2	Masters	No	1	1

Figure-7.5: Grid for calculating IDC

The last column specifies status of attribute set that either these values of attributes have already been considered or not. Second last column specifies the total number of occurrences of the current value set in dataset (so that if same value of these attributes lead to a different decision class later, we may subtract this number from current dependency value. Third last column specifies the decision class and the first n-3 columns specify the values under current attribute set. This matrix is filled for all the instances in dataset. Size of matrix will be [number of instances x {number of attributes+3}].

7.2.2 Accuracy And Efficiency of Feature Selection Algorithms using IDC

Two versions of each algorithm discussed in chapter 6 were implemented and compared with each other i.e. using positive region and using IDC. So GA was compared with GA(IDC), QR was compared with GA(IDC) as so on.

The tables 7.3 to 7.9 given below show the results of the experiments.

Table-7.3: comparison between QR and QR(IDC)

Dataset	Instances	Attributes	QR(P) Total Reducts	Memory used (MB)	QR(IDC) Total Reducts	Memory used (MB)	X times faster	% Decrease in Time
Gisette	6000	5000	9	137.37	9	114.48	97.67x faster	98%
Isolet	7797	617	2	231.96	2	18.35	5.32x faster	81.20%
Musk-2	6598	168	4	166.13	4	4.27	4.90x faster	79.60%
Egg-Eye-style	14980	15	4	856.12	4	0.97	2.83x faster	64.70%
Internet advertisement	3279	1558	89	40.04	89	19.51	3.24x fater	69.14%
UIIndoorlock	1122	529	3	4.72	3	2.25	9.0 x faster	88.88%

Table-7.4: comparison between GA and GA(IDC)

Dataset	Instances	Attributes	GA(P) Total Reducts	Memory used (MB)	GA(IDC) Total Reducts	Memory used (MB)	X times faster	% Decrease in Time
Gisette	6000	2	9	137.37	9	114.48	5.98x faster	83.28%
Isolet	7797	617	4	231.96	4	18.35	3.07 x faster	67.5%
Musk-2	6598	2	4	166.13	4	4.27	4.03 x faster	75.22%
Egg-Eye-style	14980	2	4	856.12	4	0.97	2.31 x faster	56.84%
Internet advertisement	3279	2	89	40.04	89	19.51	8.68 x faster	88.48%
UIIndoorloc	1122	529	3	4.72	3	2.25	16.0 x faster	93.75%

Table-7.5: comparison between PSO-QR and PSO-QR(IDC)

Dataset	Instances	Attributes	PSO-QR Total Reducts	Memory used (MB)	PSO-QR Total Reducts	Memory used (MB)	X times faster	% Decrease in Time
Gisette	6000	5000	13	137.37	15	114.48	1.9 x faster	46%
Isolet	7797	617	10	231.96	9	18.35	4 x faster	75%
Musk-2	6598	168	15	166.13	12	4.27	3.9 x faster	74%
Egg-Eye-style	14980	15	9	856.12	15	0.97	7.4 x faster	86.5%
Internet advertisement	3279	1558	105	40.04	99	19.51	6.6 x faster	84.9%
UIIndoorloc	1122	529	7	4.72	15	2.25	4.5 x faster	77.6%

Table-7.6: comparison between IFSA and IFSA(IDC)

Dataset	Instances	Attributes	IFSA Total Reducts	Memory used (MB)	IFSA(IDC) Total Reducts	Memory used (MB)	X times faster	% Decrease in Time
Gisette	6000	5000	15	137.37	15	114.48	2.2 x faster	53.6%
Isolet	7797	617	10	231.96	10	18.35	3.5 x faster	71.1%
Musk-2	6598	168	10	166.13	10	4.27	4.1 x faster	75.8%
Egg-Eye-style	14980	15	4	856.12	4	0.97	3.3 x faster	69.8%
Internet advertisement	3279	1558	80	40.04	80	19.51	1.8 x faster	44.4%
UIIndoorloc	1122	529	2	4,72	2	2.25	1.8 x faster	44.4%

Table-7.7: comparison between ReverseReduct and ReverseReduct(IDC)

Dataset	Instances	Attributes	RevRed Total Reducts	Memory used (MB)	RevRed(IDC)Total Reducts	Memory used (MB)	X times faster	% Decrease in Time
Gisette	6000	5000	59	137.37	59	114.48	1.6 x faster	38.2%
Isolet	7797	617	3	231.96	3	18.35	7 x faster	85.7%
Musk-2	6598	168	8	166.13	8	4.27	324.1 x faster	99.7%
Egg-Eye-style	14980	15	6	856.12	6	0.97	3.6 x faster	72%
Internet advertisement	3279	1558	795	40.04	795	19.51	1.6 x faster	38.1%
UIIndoorloc	1122	529	127	4.72	127	2.25	1.2 x faster	18.5%

Table-7.8: comparison between FSA and FSA(IDC)

Dataset	Instances	Attributes	IFSA Total Reducts	Memory used (MB)	IFSA(IDC) Total Reducts	Memory used (MB)	X times faster	% Decrease in Time
Gisette	6000	5000	41	137.37	41	114.48	1.62 x faster	38.22%
Isolet	7797	617	10	231.96	10	18.35	3.2 x faster	68.73%
Musk-2	6598	168	10	166.13	10	4.27	2.5 x faster	60.03%
Egg-Eye-style	14980	15	11	856.12	11	0.97	3.42 x faster	70.74%
Internet advertisement	3279	1558	138	40.04	138	19.51	2.79 x faster	64.14%
UIIndoorloc	1122	529	6	4.72	6	2.25	2.36 x faster	57.62%

Table-7.9: comparison between RS-IHS-QR and RS-IHS-QR(IDC)

Dataset	Instances	Attributes	IFSA Total Reducts	Memory used (MB)	IFSA(IDC) Total Reducts	Memory used (MB)	X times faster	% Decrease in Time
Gisette	6000	5000	263	137.37	263	114.48	3.01 x faster	66.67%
Isolet	7797	617	32	231.96	32	18.35	1.14 x faster	12.38%
Musk-2	6598	168	14	166.13	11	4.27	1.4 x faster	28.35%
Egg-Eye-style	14980	15	5	856.12	5	0.97	2.81 x faster	64.47%
Internet advertisement	3279	1558	95	40.04	95	19.51	8.21 x faster	87.81%
UIIndoorloc	1122	529	25	4.72	25	2.25	1.93 x faster	48.15%

In all the seven tables (7.3 to 7.9), columns from left to right show the datasets name, total number of instances and attributes. Column four, five and six, seven show the total number of reducts produced and memory used by both competitive algorithms i.e. one using positive region and other using IDC. Column eight shows how faster the algorithm using IDC is than its counterpart whereas column nine shows % decrease in time taken by algorithm using IDC. Following section shows the effectiveness of IDC based algorithm in terms of percentage decrease in execution time, memory and accuracy.

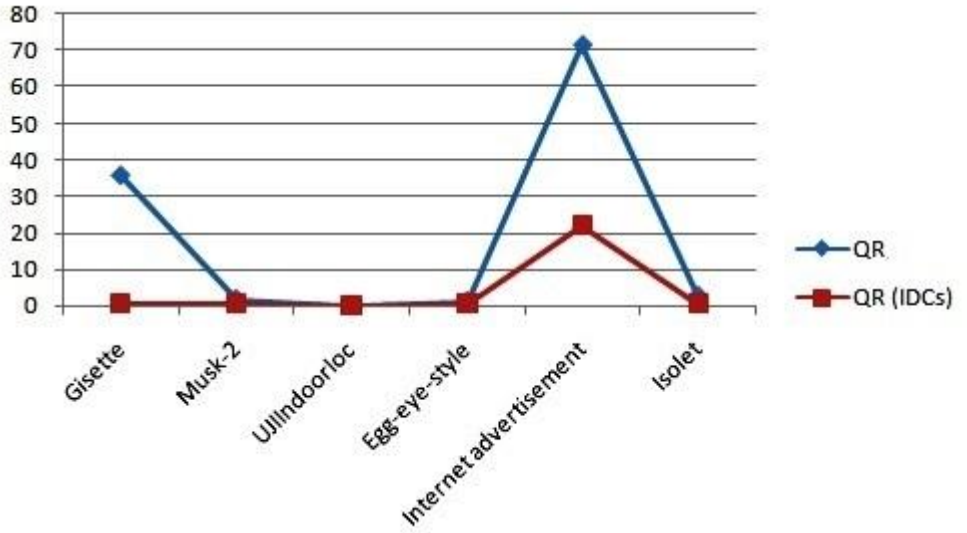


Figure-7.6: Execution time comparison b/w QR and QR(IDC)

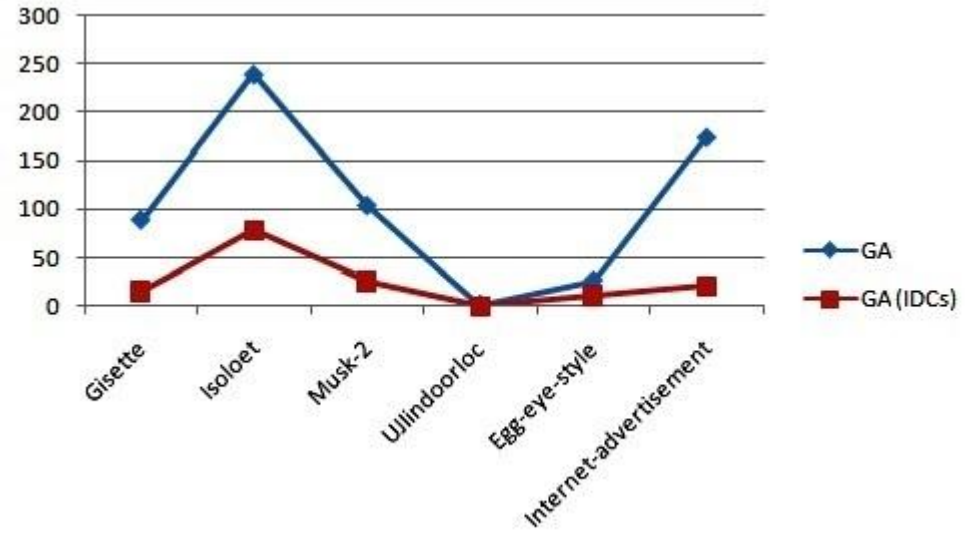


Figure-7.7: Execution time comparison b/w GA and GA(IDC)

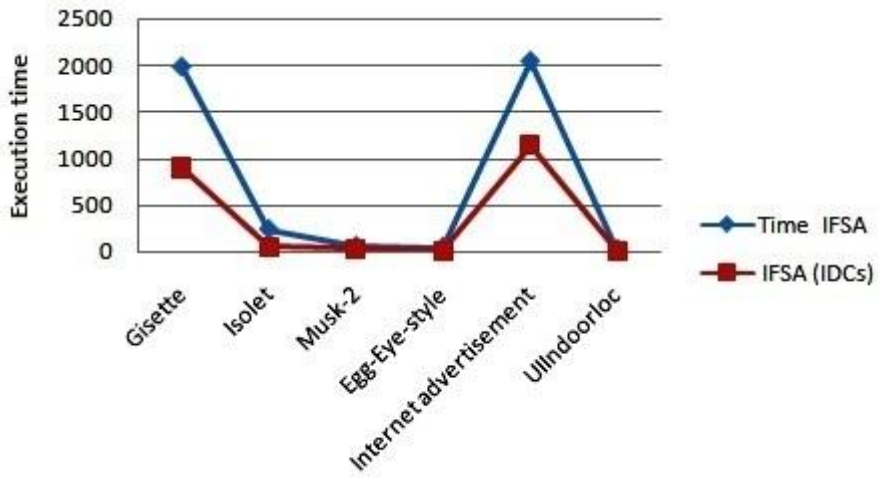


Figure-7.8: Execution time comparison between IFSA and IFSA(IDC)

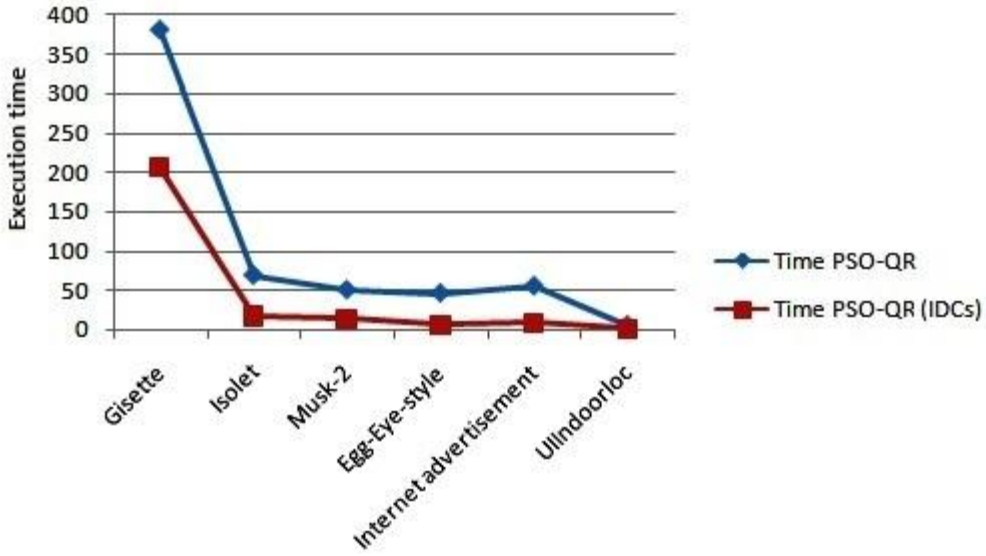


Figure-7.9: Execution time comparison between PSO-QR and PSO-QR(IDC)

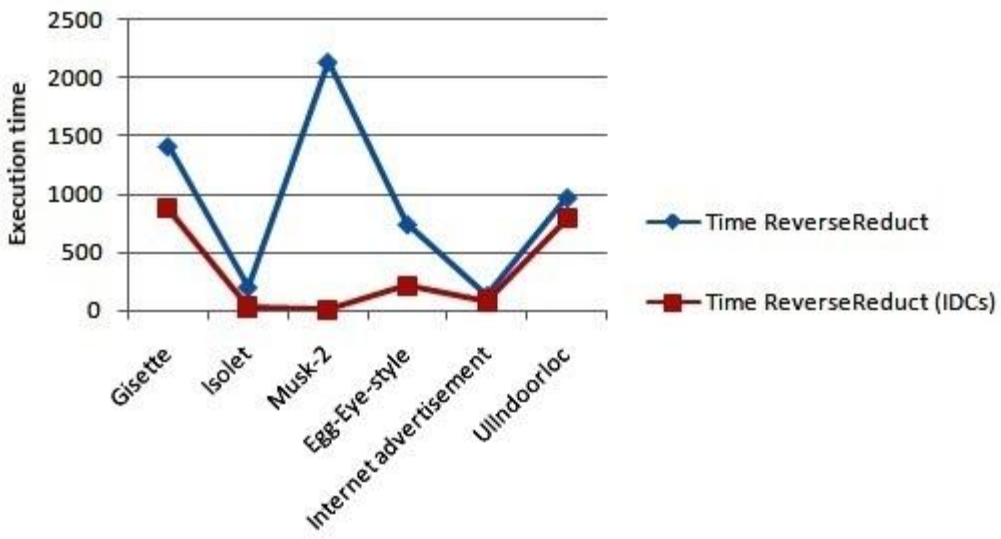


Figure-7.10: Execution time comparison between ReverseReduct and ReverseReduct(IDC)

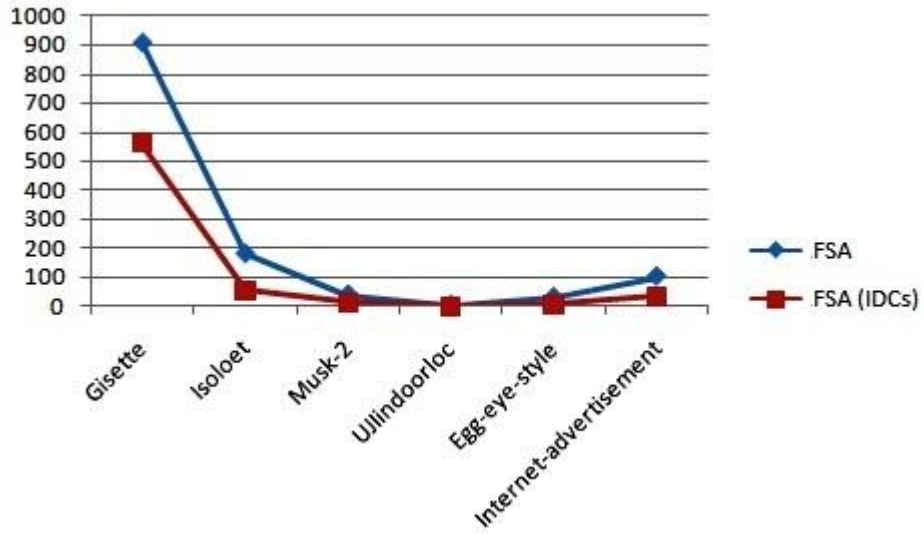


Figure-7.11: Execution time comparison between FSA and FSA (IDC)

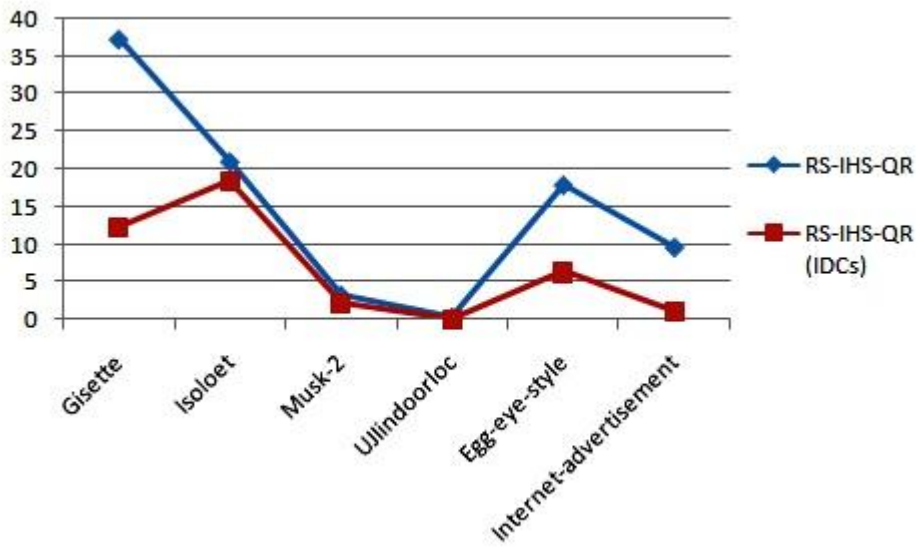


Figure-7.12: Execution time comparison between RS-IHS-QR and RS-IHS-QR (IDC)

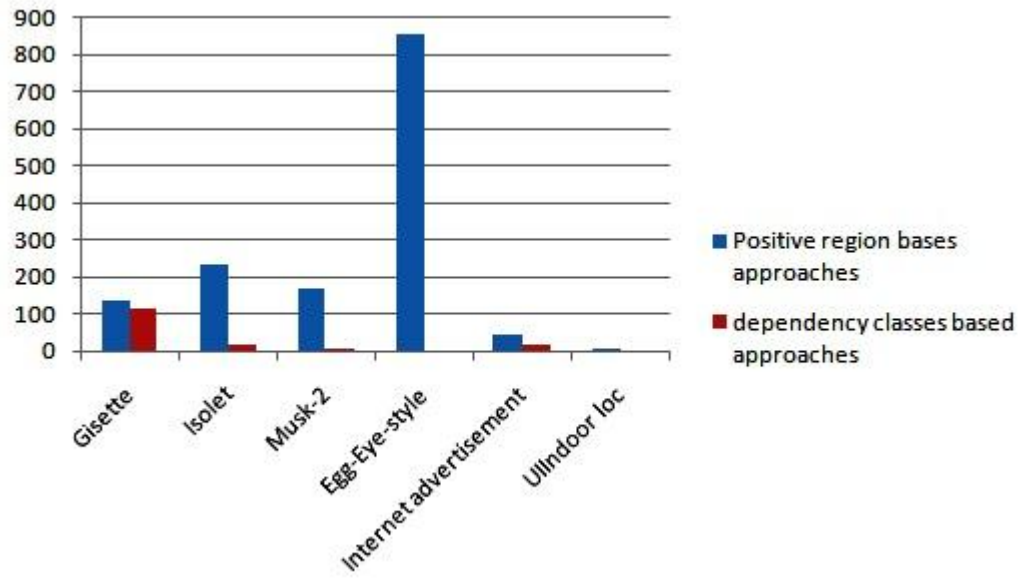


Figure-7.13: Memory comparison b/w approaches using positive region and IDC

7.2.2.1 Percentage Decrease in Execution Time

It is evident from the results that IDC based algorithms require less time as compared to their counterparts. These results support the argument that the proposed IDC can be more effective to enhance the performance of algorithms based on calculation of dependency using positive region.

Calculations show that Overall IDC based algorithms resulted in almost 65.93% decrease in execution time. The main reason behind the efficiency of IDC is that it avoids the calculation of the time consuming positive region calculation. The theory behind IDC is that it is the ratio of total number of attribute values that lead towards unique decision to total universe size. This is the point which helps reduce the execution time in case of IDC. Conventional rough set based approach on the other hand first creates equivalence class structure w.r.t. the decision attribute(s), then equivalence class structure using the attributes under consideration (on which dependency is to be measured) and finally the positive region is calculated. These steps consume too much time which makes positive region based dependency measure method totally unsuitable for other than smaller datasets.

As GA, PSO-QR, FSA and RS-IHS-QR can have different number of iterations, in different executions, based on the fitness of results, so, for these algorithms, the average execution time of single iteration was considered while keeping the other parameters as same in both cases i.e. using positive region and using IDC. Figure-7.6 to Figure-7.12 show the comparison between positive region based approaches and dependency class based approaches in graphical form.

7.2.2.2 Accuracy

Results show the equal numbers of reducts were generated both by IDC based algorithms as compared to their counterparts. The generated sets of attributes were absolutely same. However, in case of GA, PSO-QR, FSA and RS-IHS-QR, the reducts generated might be different due to their random nature, e.g. the different execution of the same GA on same set of input may generate different outputs i.e. sometimes it may find ideal solution but other executions may produce the most “close to fitness score” output after X number (generations threshold value) of generations depending on how algorithm proceeds at runtime. However, all the results were manually verified against their fitness to ensure that they 100% fulfilled the exit criteria.

7.2.2.3 Memory Usage

In our experiments (both in positive region based algorithms and those using IDC), we used intermediate data structures in their global scope. Maximum required memory was allocated at the start of algorithm and was utilized throughout to avoid run time creation/deletion of memory

(consequently to enhance performance). Results have shown that algorithms using IDC consumed less memory as compared to positive region based approach in all cases. Figure-7.13 shows the memory comparison between positive region based approaches and those using IDC in graphical form. Overall 68.4% decrease in runtime required memory was found in algorithms using IDC. However note that common data structures used by both versions of an algorithm were not considered in calculating the size of memory. For example, the size of data structure used to store fish positions was equal in both FSA and FSA(IDC), so this data structure was not considered in calculating memory size.

So, from all the above experiments we have observed that algorithms using IDC are more effective in terms of decreasing the execution time and memory but still do not compromise on the accuracy. IDC produce same dependency value as produced by conventional positive region based approach which makes IDC more effective solution to replace positive region calculation and thus cutting down in execution time. These factors also make IDC an ideal solution to be used for larger datasets where calculating the dependency value is a time complex job. Based on our above presented experimental analysis, we can conclude that IDC are effective alternate for positive region based approaches, not only in feature selection algorithms but in any rough set based algorithm which requires calculating dependency of decision attribute(s) on conditional attributes.

7.3 Experimental analysis: DDC

To justify the efficiency and effectiveness of the proposed DDC method and algorithms using DDC, we performed detailed analysis using various datasets from UCI [26] repository. The details of the datasets are given in Table-7.10.

Table-7.10: Summary of datasets used for DDC

Dataset	Instances	Attributes	Dataset characteristics / Attribute characteristics
Chess	3196	37	Multivariate / Integer
Handwriting	1593	266	Multivariate/ Real
Optidigits	1797	65	Multivariate/ Integer
Phishing	11055	31	Multivariate/ Integer, Real
Sat	2000	37	Multivariate, Sequential, Time-Series / Integer, Real
Vehicle	846	19	Multivariate/ Categorical, Integer, Real

7.3.1 Efficiency And Accuracy of DDC

Table 7-11 shows the results of experiment. First two columns specify dataset name, instances and number of attributes in each dataset. Third and fourth columns specify attribute set name and number of attributes in it. Fifth, sixth, seventh and eighth, ninth and tenth specify dependency value, time taken and memory used for DDC based approach and positive region based method respectively. Eleventh and twelfth column specify percentage decrease in execution time and percentage decrease in memory taken by DDC based method.

Table-7.11: Conventional positive region based approach vs DDC

Dataset	Inst/ Att	Attr. Set ID	Attr. Set Size	Dep(DDC)			Dep(P)			% dec in time	% dec in memory
				Dep	Time (s)	Mem (MB)	Dep	Time (s)	Mem (MB)		
Chess	3196/37	CH_1	10	0.121	0.62	0.475	0.121	0.529	38.9	88.3	98.8
		CH_2	20	0.467	0.624	0.475	0.467	2.340	38.9	73.3	98.8
		CH_3	30	0.751	1.576	0.475	0.751	3.354	38.9	53	98.8
Handwriting	1593/266	HND_1	80	1	0.748	0.814	1	1.217	9.6	38.5	91.5
		HND_2	160	1	0.748	0.814	1	1.280	9.6	41.6	91.5
		HND_3	240	1	0.765	0.814	1	1.435	9.6	46.7	91.5
Optidigits	1797/65	OPT_1	20	1	0.734	0.459	1	1.622	12.3	54.7	96.3
		OPT_2	40	1	0.811	0.459	1	1.669	12.3	51.4	96.3
		OPT_3	60	1	0.749	0.459	1	1.654	12.3	54.7	96.3
Phishing	11055/31	PHI_1	10	0.393	0.500	1.476	0.393	9.407	466.2	94.7	99.7
		PHI_2	20	0.833	2.808	1.476	0.833	13.713	466.2	79.5	99.7
		PHI_3	30	0.967	6.84	1.476	0.967	19.344	466.2	64.6	99.7
Landsat-satlite	2000/37	LND_1	5	0.957	0.531	0.297	0.957	1.451	15.2	63.4	98
		LND_2	15	1	0.593	0.297	1	1.529	15.2	61.2	98
		LND_3	30	1	0.593	0.297	1	1.622	15.2	63.4	98
vehicle	846/19	VEH_1	5	1	0.109	0.067	1	0.390	2.73	72.1	97.5
		VEH_2	10	1	0.125	0.067	1	0.421	2.73	70.3	97.5
		VEH_3	15	1	0.125	0.067	1	0.375	2.73	66.7	97.5

7.3.1.1 Percentage Decrease in Execution Time

Experiments conducted using 18 attribute sets have shown that Dep(DDC) reduces the execution time almost by 63% as compared to positive region based approach Dep(P). The basic reason behind is that Dep(DDC) directly calculates dependency thus lets us avoid the time consuming positive region calculation. Dep(DDC) only needs to scan each record to update its INSTANCECOUT AND CLASSSTATUS. After scanning complete dataset it simply calculates dependency based on uniqueness/non-uniqueness of each class. On the other hand Dep(P) requires three complex time consuming steps to calculate dependency. Firstly it computes equivalence class structure using decision attribute(s); secondly it requires equivalence class structure using the conditional attributes and finally it computers positive region on the base of which dependency is calculated. All these steps make Dep(P) too time consuming to be used for feature subset selection.

7.3.1.2 Memory Usage

Dep(DDC) consumes less memory than Dep(P). Results have shown that Dep(DDC) reduced the required runtime memory almost by 96% for eighteen datasets with different number of attributes and records. The reason behind is that Dep(DDC) does not require to calculate equivalence class structure as required by the first two steps of Dep(P). To calculate these class structures we require substantial amount of memory. To calculate the equivalence class structure for decision attribute we need memory of size calculated as:

M([X]D) = Number of decision classes * (maximum number of records in any class+3)

The memory will be used in the form of two dimensional matrix having number of rows equal to number of decision classes and number of columns equal to maximum number of records in any class plus three extra columns. Note that three extra columns are for control purpose, they will contain decision attribute value, total number of objects in current class and index of last object in current row. In table-3-1(b) there are three decision classes (i.e. Platinum, Gold and Silver) and "Gold" class has maximum number (three) of records in it. So, the size of matrix required to calculate $[X]_D$ will be:

$M([X]_D) = \{3,6\}$

If memory taken by one matrix element is 4 bytes, then the total memory required to calculate $[X]_D$ will be:

$M([X]_D) = 3*(6)*4 = 52$ bytes.

The matrix will have runtime contents shown by Figure-7.14:

Platinum	2	5	X1	X2	
Gold	3	6	X3	X5	X7
Silver	2	5	X4	X6	

Figure-7.14: Runtime Equivalence class structure $[X]_D$

Similarly to calculate $[X]_C$, we require a two dimensional matrix having rows equal to number of records in dataset and columns equal to number of records plus three extra columns. Note that extra three columns are gain for control purpose. So the memory required by [X]C in our case will be equal to:

$M([X]_D) = 7*(10)*4 = 280$ bytes.

So overall we need 323 bytes of runtime memory to calculate Dep(P).

To calculate Dep(DDC) we need only a single grid as discussed in section-3. The Grid is again a two dimensional matrix with following dimensions:

No. of rows = No. of records in dataset

No. of Columns = number of conditional attributes + number of decision attributes + 2

In our example:

No. of Rows = 7

No. of Columns = 2 +1 + 2 = 5

So, required memory = 7*5*4 = 140 bytes.

It is clear from above example that Dep(DDC) takes almost 50% less memory as compared to Dep(P).

7.3.1.3 Accuracy

It is clear from Table-7.11 that Dep(DDC) shows same accuracy as that shown by conventional positive region based approach. The reason behind is that Dep(DDC) calculates the same unique/non-unique classes that represent positive region. However, instead of using equivalence class structure and calculating positive region, it directly determines these unique/non-unique classes based on the decision class the values of attributes lead to.

From the above measures it is clear that Dep(DDC) is more effective and accurate as compared to Dep(P) and can safely be used in any of the feature selection algorithm.

7.3.2 Efficiency And Accuracy of Algorithms using DDC

Experimental analysis has shown that algorithms using DDC based approach have been more effective both in terms of percentage decrease in execution time and memory still maintaining the accuracy. Table-7.12 to Table-7.16 show the results of the analysis. First three columns in each table provide dataset name, number of instances and number of attributes. Columns four, five and six, seven show the number of attributes in reduct and memory used by DDC based approach and its counterpart using positive region respectively. Finally columns eight and nine show the percentage decrease in execution time and memory resulted in case of DDC based algorithms.

Table-7.12: comparison between IFSA and IFSA(DDC)

Dataset	Instances	Attributes	IFSA (DDC) Total Reducts	Memory used (MB)	IFSA(P) Total Reducts	Memory used (MB)	X times faster	% Decrease in Time
Chess	3196	37	36	0.243	36	19.5	4.1	75.4
Handwriting	1593	266	28	0.817	28	4.84	2.8	64.2
Optidigits	1797	65	20	0.233	20	6.16	2.1	53.3
Phishing	11055	31	30	0.716	30	233.47	12.6	92
Sat	2000	37	20	0.164	20	7.63	1.6	38.6
Vehicle	846	19	5	0.035	5	1.36	3.8	73.7

Table-7.13: comparison between GA and IFSA(DDC)

Dataset	Instances	Attributes	IFSA Total Reducts	Memory used (MB)	IFSA(IDC) Total Reducts	Memory used (MB)	X times faster	% Decrease in Time
Chess	3196	37	36	0.243	36	19.5	22.6	95.6
Handwriting	1593	266	28	0.817	28	4.84	2.4	58.7
Optidigits	1797	65	20	0.233	20	6.16	2.8	64.4
Phishing	11055	31	30	0.716	30	233.47	24.5	95.9
Sat	2000	37	20	0.164	20	7.63	3	66.8
vehicle	846	19	5	0.035	5	1.36	2.2	54.3

Table-7.14: comparison between IHS and IHS(DDC)

Dataset	Instances	Attributes	IFSA Total Reducts	Memory used (MB)	IFSA(IDC) Total Reducts	Memory used (MB)	X times faster	% Decrease in Time
Chess	3196	37	36	0.243	36	19.5	1.7	40.3
Handwriting	1593	266	28	0.817	28	4.84	2.1	53.4
Optidigits	1797	65	20	0.233	20	6.16	1.6	39
Phishing	11055	31	30	0.716	30	233.47	3.3	69.4
Sat	2000	37	20	0.164	20	7.63	2.4	58.5
vehicle	846	19	5	0.035	5	1.36	2.1	51.8

Table-7.15: comparison between FSA and FSA (DDC)

Dataset	Instances	Attributes	IFSA Total Reducts	Memory used (MB)	IFSA(IDC) Total Reducts	Memory used (MB)	X times faster	% Decrease in Time
Chess	3196	37	36	0.243	36	19.5	1.5	33.3
Handwriting	1593	266	28	0.817	28	4.84	1.6	38.7
Optidigits	1797	65	20	0.233	20	6.16	1.9	46.6
Phishing	11055	31	30	0.716	30	233.47	6	83.2
Sat	2000	37	20	0.164	20	7.63	91.2	98.9
vehicle	846	19	5	0.035	5	1.36	1.9	47.6

Table-7.16: comparison between PSO(DDC) and PSO

Dataset	Instances	Attributes	IFSA Total Reducts	Memory used (MB)	IFSA(IDC) Total Reducts	Memory used (MB)	X times faster	% Decrease in Time
Chess	3196	37	5	0.243	4	19.5	9.8	89.7
Handwriting	1593	266	23	0.817	23	4.84	43.3	97.7
Optidigits	1797	65	12	0.233	37	6.16	12.3	91.9
Phishing	11055	31	12	0.716	18	233.47	37.6	97.3
Sat	2000	37	18	0.164	20	7.63	2.9	65.3
vehicle	846	19	12	0.035	13	1.36	1.3	22.2

7.3.2.1 Percentage Decrease in Execution Time

Experiments have shown that algorithms using DDC based method show a significant decrease in execution time. We have observed decrease of almost 95% for six datasets. This justifies our claim that algorithms using DDC method are more efficient as compared to those using positive region based approach. As PSO,IHS, GA and FSA are random in nature, they may produce results in different iterations in different runs, so to ensure the unbiased analysis we used the average time for single iteration. The main reason behind is that the efficiency of DDC method, which avoids positive region. The positive region based approaches on the other hand suffer from time consuming task of positive region calculation, while DDC based methods simply calculate dependency using number of unique/non-unique classes. Figure-7.15 to Figure-7.19 shows the execution time comparison of both versions of each algorithm.

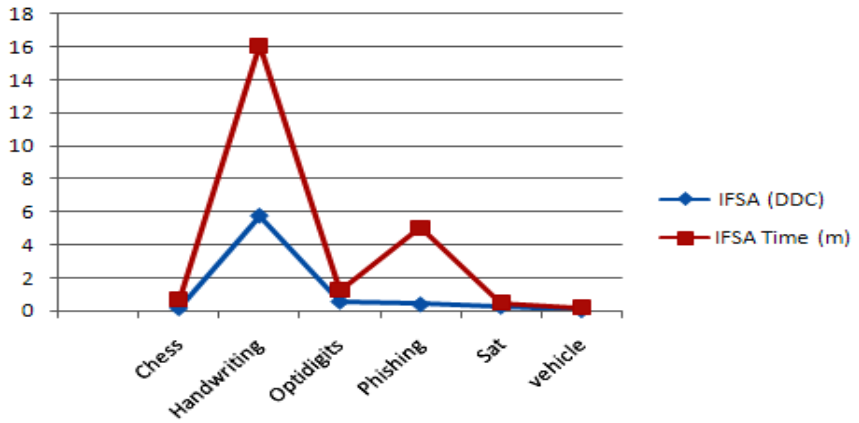


Figure-7.15: comparison of execution time b/w/ IFSA (DDC) & IFSA

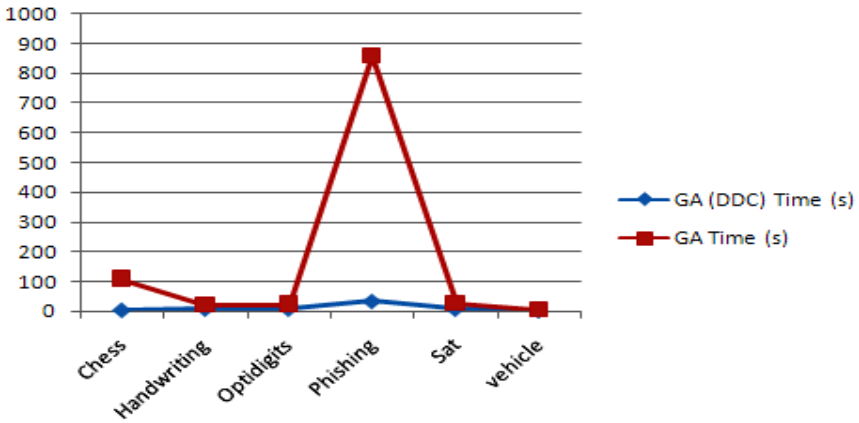


Figure-7.16: comparison of execution time b/w/ GA (DDC) & GA

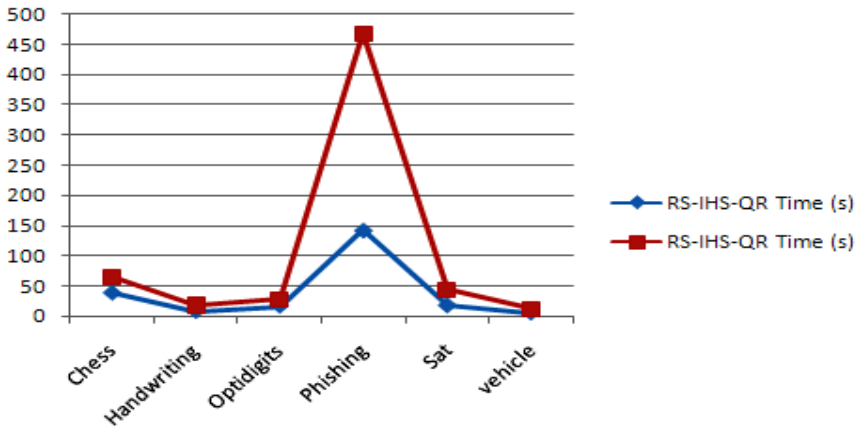


Figure-7.17: comparison of execution time b/w/ RS-IHS (DDC) & RS-IHS

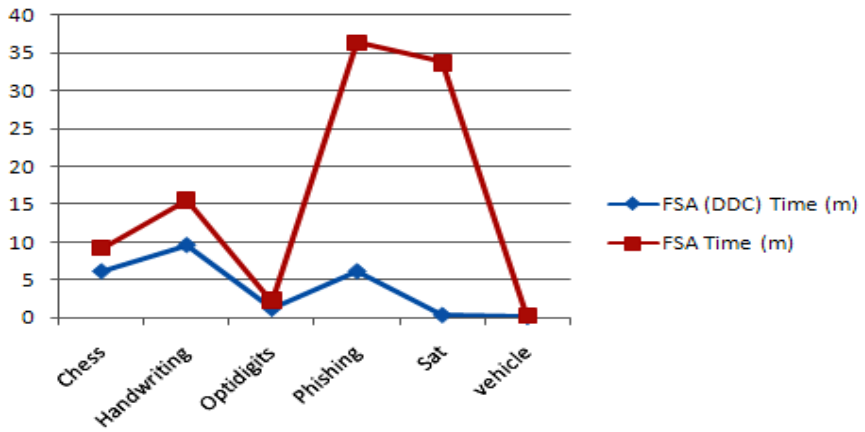


Figure-7.18: comparison of execution time b/w/ FSA (DDC) & FSA

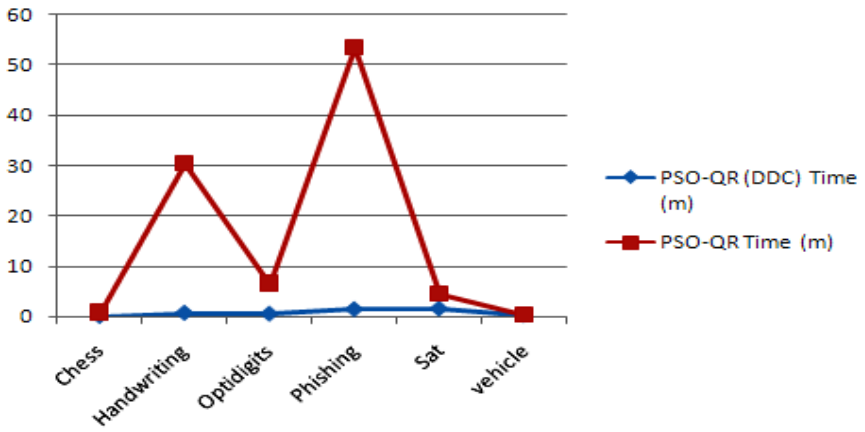


Figure-7.19: comparison of execution time b/w/ PSO-QR (DDC) & PSO

7.3.2.2 Memory Usage

Algorithms using DDC have taken less memory as compared to positive region based approaches. Results have shown that DDC based approaches have shown almost 95% decrease in memory on average for six datasets. The reason behind is that DDC based approaches require only one matrix to calculate number of unique and non-unique classes. Positive region based approaches on the other hand require two matrices to calculate equivalence class structure for first two steps as discussed in section 2.5.

7.3.2.3 Accuracy

DDC based approaches have shown same accuracy as that of conventional approach. The attributes in reduct set calculated by algorithms were different in some cases due to random nature of the

algorithms; however, the produced results were manually tested against their accuracy to be represented as candidate feature subsets and they fully qualified. This justifies our claim that DDC based approaches can successfully be used in any feature selection algorithm with absolute accuracy.

7.4 Experimental analysis: Redefined Preliminaries

To verify the proposed definitions, lower and upper approximations were calculated using both conventional and proposed method. Table-7.17 shows the result of lower and upper approximation calculated by both methods. In this table, columns from left to right are dataset name, cardinality of lower approximation using proposed definition, cardinality of lower approximation calculated using conventional method, time required to calculate lower approximation using proposed definition, time required to calculate lower approximation using conventional method, memory required in calculating lower approximation by proposed method, memory required in calculating lower approximation by conventional method, percentage decrease in execution time taken by proposed method and percentage decrease in memory required by proposed method.

Table-7.17: Conventional Lower Approximation vs Redefined Lower Approximation

Datasets	Cardinality LA(RP)	Cardinality LA(Ind)	Time (sec) LA (RP)	Time (sec) LA(Ind)	Memory (MB) LA(RP)	Memory (MB) LA(Ind)	%dec in time	%dec in memory
Vehicle	217	217	0.12	0.33	0.38	1.36	63.64	72.06
Musk1	207	207	0.05	0.1	0.34	1.46	50	76.71
Land-Sat	461	461	0.5	1.35	2.30	7.63	62.96	69.86
Handwriting	158	158	0.83	1.41	1.3	4.84	41.13	73.14
Musk2	1017	1017	1.992	6.19	7.5	83	67.82	90.96

Similarly Table-7.18 shows the results for proposed upper approximation calculation method.

Table-7.18: Conventional Upper Approximation vs Redefined Upper Approximation

Datasets	Cardinality UA(RP)	Cardinality UA(Ind)	Time (sec) UA (RP)	Time (sec) UA(Ind)	Memory (MB) UA(RP)	Memory (MB) UA(Ind)	%dec in time	%dec in memory
Vehicle	217	217	0.12	0.33	0.37	1.36	63.64	72.79
Musk1	207	207	0.04	0.120	0.34	1.46	66.67	76.71
Land-Sat	461	461	0.51	1.94	2.3	7.73	73.71	70.25
Handwriting	158	158	0.72	1.38	1.3	4.84	47.83	73.14
Musk2	1017	1017	2.25	7.312	7.5	83	69.23	90.96

7.4.1 Accuracy

For accuracy, the results were compared both in case of lower and upper approximations. Table-7.19 and Table-7.20 show the actual objects obtained by lower and upper approximations using both approaches. First column in both tables shows dataset, second column shows the approach used. For each dataset both redefined preliminaries based approach LA (RP) and indiscernibility based approach LA (Ind) was used. Third and fourth columns specify the total number of objects obtained and decision class used. Finally the fifth column specifies the actual objects obtained using each approach. In dataset objects were numbers from one to n, where n=1 represents first objects (i.e. row number one), n=2 represents second object and so on.

Table-7.19: Lower approximation: Indiscernibility Vs redefined preliminaries based approach

Dataset	Technique	No. of objects	Decision class	Objects
Vehicle	LA(RP)	207	X=1	3,10,12,19,25,27,28,30,32,39,44,45,50,51,52,57,77,78,91,93,97,106,118,121,124,131,132,133,139,141,149,151,154,159,163,164,167,168,181,184,185,193,195,197,202,217,225,227,229,230,232,234,244,248,250,256,257,259,261,262,265,268,272,279,284,286,290,298,299,301,307,311,318,321,324,325,330,336,343,347,352,355,358,361,362,363,366,368,377,378,379,387,395,401,408,410,411,420,423,429,431,433,435,440,441,447,455,460,469,476,481,488,491,492,504,506,507,514,518,519,521,523,526,527,531,533,537,543,544,550,553,555,558,560,563,566,567,568,571,572,576,577,583,584,594,600,604,606,615,621,623,624,625,626,631,637,643,648,649,650,651,652,655,660,662,663,664,668,675,689,690,691,693,695,697,702,706,712,713,714,717,720,722,727,737,741,744,750,751,754,757,758,762,765,767,770,777,779,

				781,784,787,789,790,798,807,808,810,818,820,821,833,834,835,838,842,844,845
	LA(Ind)	207	X=1	3,10,12,19,25,27,28,30,32,39,44,45,50,51,52,57,77,78,91,93,97,106,118,121,124,131,132,133,139,141,149,151,154,159,163,164,167,168,181,184,185,193,195,197,202,217,225,227,229,230,232,234,244,248,250,256,257,259,261,262,265,268,272,279,284,286,290,298,299,301,307,311,318,321,324,325,330,336,343,347,352,355,358,361,362,363,366,368,377,378,379,387,395,401,408,410,411,420,423,429,431,433,435,440,441,447,455,460,469,476,481,488,491,492,504,506,507,514,518,519,521,523,526,527,531,533,537,543,544,550,553,555,558,560,563,566,567,568,571,572,576,577,583,584,594,600,604,606,615,621,623,624,625,626,631,637,643,648,649,650,651,652,655,660,662,663,664,668,675,689,690,691,693,695,697,702,706,712,713,714,717,720,722,727,737,741,744,750,751,754,757,758,762,765,767,770,777,779,781,784,787,789,790,798,807,808,810,818,820,821,833,834,835,838,842,844,845
Musk-2	LA(RP)	1017	X=1	Object no. 1 to Object no. 1017 (i.e. first 1017 objects)
	LA(Ind)	1017	X=1	Object no. 1 to Object no. 1017 (i.e. first 1017 objects)
Handwriting	LA(RP)	158	X=1	180-199, 379-398, 579-597, 778-796, 1156-1195, 1554-1593
	LA(Ind)	158	X=1	180-199, 379-398, 579-597, 778-796, 1156-1195, 1554-1593
Land-sat	LA(RP)	461	X=1	915,941,962,981-984, 1004-1008, 1023-1025, 1042-1047, 1066-1069, 1085-1090, 1112-1115, 1131-1134, 1156-1163, 1185-1189, 1205-1211, 1229-1234, 1254-1256, 1273-1279, 1299-1304, 1323-1329, 1350-1355, 1375-1383, 1392-1404, 1417-1428, 1441-1452, 1461-1477, 1485-1495, 1510-1524, 1534-1549, 1556-1572, 1580-1587, 1597-1605, 1613-1623, 1625,1626,1636-1648, 1659-1668, 1671,1683-1694, 1704-1709, 1729-1750, 1759-1774, 1791-1804, 1815-1829, 1845-1859, 1873-1893, 1904-1917, 1930-1948, 1955-1972, 1979-1997
	LA(Ind)	461	X=1	915,941,962,981-984, 1004-1008, 1023-1025, 1042-1047, 1066-1069, 1085-1090, 1112-1115, 1131-1134, 1156-1163, 1185-1189, 1205-1211, 1229-1234, 1254-1256, 1273-1279, 1299-1304, 1323-1329, 1350-1355, 1375-1383, 1392-1404, 1417-1428, 1441-1452, 1461-1477, 1485-1495, 1510-1524, 1534-1549, 1556-1572, 1580-1587, 1597-1605, 1613-1623, 1625,1626,1636-1648, 1659-1668, 1671,1683-1694, 1704-1709, 1729-1750, 1759-1774, 1791-1804, 1815-1829, 1845-1859, 1873-1893, 1904-1917, 1930-1948, 1955-1972, 1979-1997
Musk-1	LA(RP)	207	X=1	Object 1 to object 207 (first 207 objects)
Musk-1	LA(Ind)	207	X=1	Object 1 to object 207 (first 207 objects)

Table-7.20: Upper approximation: Indiscernibility Vs redefined preliminaries based approach

Dataset	Technique	No. of objects	Decision class	Objects
Vehicle	LA(RP)	217	X=1	3,10,12,19,25,27,28,30,32,39,44,45,50,51,52,57,77,78,91,93,97,106,118,121,124,131,132,133,139,141,149,151,154,159,163,164,167,168,181,184,185,193,195,197,202,217,225,227,229,230,232,234,244,248,250,256,257,259,261,262,265,268,272,279,284,286,290,298,299,301,307,311,318,321,324,325,330,336,343,347,352,355,358,361,362,363,366,368,377,378,379,387,395,401,408,410,411,420,423,429,431,433,435,440,441,447,455,460,469,476,481,488,491,492,504,506,507,514,518,519,521,523,526,527,531,533,537,543,544,550,553,555,558,560,563,566,567,568,571,572,576,577,583,584,594,600,604,606,615,621,623,624,625,626,631,637,643,648,649,650,651,652,655,660,662,663,664,668,675,689,690,691,693,695,697,702,706,712,713,714,717,720,722,727,737,741,744,750,751,754,757,758,762,765,767,770,777,779,781,784,787,789,790,798,807,808,810,818,820,821,833,834,835,838,842,844,845
	LA(Ind)	217	X=1	3,10,12,19,25,27,28,30,32,39,44,45,50,51,52,57,77,78,91,93,97,106,118,121,124,131,132,133,139,141,149,151,154,159,163,164,167,168,181,184,185,193,195,197,202,217,225,227,229,230,232,234,244,248,250,256,257,259,261,262,265,268,272,279,284,286,290,298,299,301,307,311,318,321,324,325,330,336,343,347,352,355,358,361,362,363,366,368,377,378,379,387,395,401,408,410,411,420,423,429,431,433,435,440,441,447,455,460,469,476,481,488,491,492,504,506,507,514,518,519,521,523,526,527,531,533,537,543,544,550,553,555,558,560,563,566,567,568,571,572,576,577,583,584,594,600,604,606,615,621,623,624,625,626,631,637,643,648,649,650,651,652,655,660,662,663,664,668,675,689,690,691,693,695,697,702,706,712,713,714,717,720,722,727,737,741,744,750,751,754,757,758,762,765,767,770,777,779,781,784,787,789,790,798,807,808,810,818,820,821,833,834,835,838,842,844,845

Musk-2	LA(RP)	1017	X=1	Object no. 1 to Object no. 1017 (i.e. first 1017 objects)
	LA(Ind)	1017	X=1	Object no. 1 to Object no. 1017 (i.e. first 1017 objects)
Handwriting	LA(RP)	158	X=1	180-199, 379-398, 579-597, 778-796, 1156-1195, 1554-1593
	LA(Ind)	158	X=1	180-199, 379-398, 579-597, 778-796, 1156-1195, 1554-1593
Land-sat	LA(RP)	461	X=1	915,941,962,981-984, 1004-1008, 1023-1025, 1042-1047, 1066-1069, 1085-1090, 1112-1115, 1131-1134, 1156-1163, 1185-1189, 1205-1211, 1229-1234, 1254-1256, 1273-1279, 1299-1304, 1323-1329, 1350-1355, 1375-1383, 1392-1404, 1417-1428, 1441-1452, 1461-1477, 1485-1495, 1510-1524, 1534-1549, 1556-1572, 1580-1587, 1597-1605, 1613-1623, 1625,1626,1636-1648, 1659-1668, 1671,1683-1694, 1704-1709, 1729-1750, 1759-1774, 1791-1804, 1815-1829, 1845-1859, 1873-1893, 1904-1917, 1930-1948, 1955-1972, 1979-1997
	LA(Ind)	461	X=1	915,941,962,981-984, 1004-1008, 1023-1025, 1042-1047, 1066-1069, 1085-1090, 1112-1115, 1131-1134, 1156-1163, 1185-1189, 1205-1211, 1229-1234, 1254-1256, 1273-1279, 1299-1304, 1323-1329, 1350-1355, 1375-1383, 1392-1404, 1417-1428, 1441-1452, 1461-1477, 1485-1495, 1510-1524, 1534-1549, 1556-1572, 1580-1587, 1597-1605, 1613-1623, 1625,1626,1636-1648,1659-1668,1671,1683-1694, 1704-1709, 1729-1750, 1759-1774, 1791-1804, 1815-1829, 1845-1859, 1873-1893, 1904-1917, 1930-1948, 1955-1972, 1979-1997
Musk-1	LA(RP)	207	X=1	Object no. 1 to object no. 207
	LA(Ind)	207	X=1	Object no. 1 to object no. 207

Results have shown that proposed definitions provide the same results for same concepts as produced by conventional method. The reason behind is that proposed definitions are semantically same as conventional definitions however, computationally they are efficient because the computationally complex and expensive step of calculating equivalence classes is skipped. Instead the lower and upper approximations are calculated directly by calculating the classes that lead to

same decision class (in case of lower approximation) or different decision class (in case of upper approximation). Analysis has shown that proposed redefinitions have produced same results as given by conventional definitions i.e. both the proposed redefinitions and the conventional methods have given same objects for same concepts.

7.4.2 Percentage Decrease in Execution Time

For percentage decrease in execution time, system stopwatch was used. It was found that proposed redefinitions have shown 57.1% decrease in execution time for redefined lower approximation and 64.2% decrease in case of redefined upper approximation, for five publically available datasets. The reason behind is that proposed redefinitions do not require equivalence classes and calculate the approximations in single step. The entire dataset is scanned and the objects leading to same or different decision class for same value of attributes are calculated. On the other hand, in case of conventional definitions, three steps are involved.

7.4.3 Memory Usage

Similarly required runtime memory was compared both in case of conventional definitions and proposed redefinitions. It was found that for five datasets, proposed redefinitions have shown 76.5% decrease in required runtime memory. For this purpose major data structures used were calculated. We used two dimensional arrays (Grids) for both approaches. In case of conventional approach, the size of grid used was larger than the one used for proposed definitions. The reason behind is that in conventional case the grid requires to store all the objects in the form of equivalence class structure (every object belongs to one equivalence class), whereas in case of proposed redefinitions we only need to store the objects that belong to the concept under consideration.

So, for conventional definition of lower approximation, size of grid will be:
Size of Grid= [maximum no. of rows x (maximum no. of rows +1)] x datatype size
Note that the extra attribute (column in grid) is used for control purpose.
For proposed redefinitions, on the other hand, size of required grid is:
Size of Grid = [(no. of attribute + maximum number of objects in concept + 3) x (maximum no. of rows)] x datatype size.

Again, extra three columns are for control purpose.

Here we need lesser memory because normally the total number of attributes and number of objects belonging to any concept are lesser than total number of objects in dataset.

Here we explain it with an example. We will consider the "Vehicle" dataset take from [2] for this purpose. This dataset comprises of 846 objects and 18 attributes (excluding decision class), decision class (concept) D=1 contains maximum 217 objects.

So, memory required in calculating lower approximation using conventional approach:

Size of Grid= [maximum no. of rows x (maximum no. of rows +1)] x datatype size

= (846*847)*2 = 1399.535156 Bytes = 1.36MB

Similarly, memory required using proposed redefinitions will be:

Size of Grid = [(no. of attribute + maximum number of objects in concept + 3) x (maximum no. of rows)] x datatype size

Size of Grid = [(18 + 217+ 3) x (846)] x 2 = 393.2578125 bytes = 0.38MB

7.5 Experimental analysis: Redefined Preliminaries Based Feature Selection

To justify the proposed algorithm, it was compared with four state of the art algorithms using conventional indiscernibility based dependency measure i.e. PSO-QR(Ind) [22], GA(Ind) [24], IFSA (Ind) [25], AFSA (Ind) [26] . Algorithms were executed using five publicly available datasets and results were compared. For calculating decrease in execution time, system stopwatch was used. It was started after reading the dataset and was stopped after output was generated. Table-7.21 shows the results of the experiments.

Table-7.21: RPFS vs conventional indiscernibility based approaches

		RPFS		PSO-QR [22]		GA [24]		IFSA [25]		AFSA [26]	
	Records / attributes	**Reducts**	**Time (m)**	**Reducts**	**Time (m)**	**Reducts**	**Time (m)**	**Reducts**	**Time (m)**	**Reducts**	**Time (m)**
Vehicle	**846/19**	4	0.07	8	0.44	7	0.11	5	0.138	13	0.2
Musk2	**6598/168**	18	5.18	81	237.5	79	6.34	20	57.5	78	59.3
handwriting	**1593/266**	9	0.33	130	24.1	123	0.51	28	16.13	133	8.43
Land-sat	**2000/36**	10	0.32	15	3.5	14	0.48	20	0.43	18	1.31
Musk1	**476/168**	2	0.1	136	1.43	19	0.1	20	1.19	121	1.6

Results were analyzed on two parameters i.e. accuracy and execution runtime.

Accuracy specifies the appropriateness of set of features selected as final feature subset. Two aspects were considered in this regard i.e. the size of feature subset selected and the dependency of selected subset. It was observed that the size of feature subset selected by proposed algorithm was always lesser than competitive algorithms. The reason behind is that proposed algorithm selects features on the basis of dependency value. So the feature with highest dependency i.e. the features having more information are selected first which results in optimal feature subset. Dependency of selected feature subsets was manually verified by calculating dependency using conventional method to further ensure the accuracy. It was found that resulted features had dependency equal to entire dataset which means that resulted subset was absolutely appropriate candidate solution.

Results have shown that proposed solution is more efficient. It is observed that proposed solution show a significant decrease in execution time.

The reason behind is that it only needs to check the dependency of each individual attribute and then select the attributes with higher dependency to find out the optimal feature subset. Selecting the attributes with higher degrees of dependency result in minimal number of combinations to be tested to find the final subset. We have already discussed that combinations of features with higher degree of dependency are likely to increase dependency value (of overall combination) more than combinations of features of lower degree dependency. So, using this technique lets us find the feature subset in very fewer attempts after dependency of each individual attribute is calculated, thus algorithm results the output feature subset more efficiently as compared to other algorithms.

Furthermore, to calculate dependency we have used the proposed lower approximation redefinitions rather than using the conventional dependency method, which substantially adds to efficiency of algorithm. Results (given in Table-10) have shown that for five publicly available datasets proposed algorithm showed 68.238% decrease in execution time on average.

By enhancing the efficiency with optimal feature subset, the proposed algorithm can save substantial amount of time which can be utilized in further tasks e.g. classification, clustering, rule extraction etc. Since the resulted feature subset has minimum number of attributes, so using proposed algorithm as pre-processor can enhance the efficiency of other tasks which take this feature subset as input.

7.6 Summary

A comparison framework was designed to perform experimental analysis. The framework comprised of three components, percentage decrease in execution time, memory used and accuracy. To justify the proposed solution, at first step both IDC and DDC based methods were analyzed. In second step, feature selection algorithms based on IDC and DDC were tested. The experimental results have shown that algorithms using IDC and DDC were more effective than their counterparts using the positive region-based approach in terms of accuracy, execution time and required runtime memory.

Chapter 8: Conclusion and Future work

The main objective of the current research was to propose alternate methods for calculating rough set based Lower Approximation, Upper Approximation and Dependency measure. We have then proposed new feature selection algorithm using these measures. The conventional methods for calculating these measures are computationally too expensive to be used for performing feature selection on large datasets. Experimental analysis have shown the significance of proposed approaches both in terms of efficiency and effectiveness. In this chapter we will provide the overall summary of our research work and will provide some insight into its future extension.

8.1 Lower And Upper Approximations

Conventional Rough Set Based Lower and Upper approximations use equivalence class structure to calculate approximations which is computationally expensive job. However, on the concept of lower approximation provided by RST, we have proposed a new definition as follows:

$\underline{C}X = \{\forall\, x \in U, c\ \in C,\ a \neq b |\ x_{\{c \cup d\}} \rightarrow a,\ x_{\{c \cup d\}} \nrightarrow b\}$

i.e. the lower approximation of concept "X" w.r.t. the attribute set "c", is set of objects such that for each occurrence of the object, the same value of conditional attribute "c" always leads to the same decision class value. So, if there are "n" occurrences of an object, then all of them lead to same decision class (for same value of attributes), which alternatively means that for a specific value of an attribute, we can with surety say that object belongs to a certain decision class. This is exactly equal to conventional definition of lower approximation.

Similarly the following definition of upper approximation was proposed:

$$\overline{C}X = \underline{C}X \cup \{\forall\, x \in U, c\ \in C,\ a \neq b |\ x_{\{c\}} \rightarrow a,\ x'_{\{c\}} \rightarrow b\}$$

This definition will be read as follows:

Provided that that objects x and x' are indiscernible wr.t. to attribute(s) c, they will be part of an upper approximation if either they belong to lower approximation or at least one of their occurrences leads to decision class belonging to concept X. So objects x and x' belong to upper approximation if both occurrences of them lead to different decision class for the same value of attributes. Results have shown that the proposed heuristics based approach provide significant increase in efficiency and performance without affecting accuracy.

8.2 Dependency Classes

Majority of feature selection algorithms use rough set based dependency measure for feature selection. However, using conventional concept of dependency provided by rough set is computationally expensive approach. It uses positive region for calculating dependency which involves three steps i.e. calculating equivalence class structure for decision class, calculating equivalence class for conditional attributes and finally calculating positive region. The complex computation involved in positive region based approach makes it inappropriate for algorithms performing feature selection for larger datasets. To overcome the issue two alternate methods were proposed based on dependency classes. A dependency class is a rule that defines how degree of dependency of decision class "D" on conditional attribute "C" changes as we read new record in dataset.

8.2.1 Incremental Dependency Classes

Incremental dependency classes are set of four rules that govern, how dependency value changes with each new record.

They are:

- Existing boundary region class: If same value of attribute leads to different decision classes, it decreases the dependency
- Positive region class: If adding a record, does not lead to a different decision class for same value of that attribute, dependency will increase.
- Initial positive region class: If the value appears in the data set for the first time for that attribute, then dependency increases.
- Boundary region class: If same value (which was leading to unique decision previously) of attribute leads to different decision, then adding the new record reduces the dependency.

8.2.2 Direct Dependency Classes

Direct dependency classes specify how dependency value change as new record is read in dataset. For a decision class D, the dependency K of D on C is as shown in table 8-1.

Table-8.1: How DDC calculates dependency

Dependency	No of unique/non-unique classes
0	If there is no unique class
1	If there is no non-unique class
N	Otherwise where $0 < n < 1$

It means that reading a record belonging to unique class will increase the dependency and reading a record belonging to non-unique class will decrease dependency.

8.3 Feature Selection Using Dependency Classes

On the basis of the proposed incremental and direct dependency classes, various feature selection algorithms, originally using positive region based dependency measure were re-implemented. In these algorithms, step of calculating positive region based dependency measure was replaced with dependency classes based methods.

8.4 Experimental Analysis

Experiments were performed to justify the significance of proposed solutions. Experiments were carried out in two steps. In first step accuracy and efficiency of dependency classes themselves was analyzed. Results have shown the significance of dependency classes by considerably reducing the execution time and runtime memory utilization while still not compromising accuracy. On average, IDC reduced the execution time by 54.5% for 18 attribute sets. The reason behind is that the IDC have successfully avoided the complex computations of calculating positive regions. Memory taken by the IDC was less than positive region based approach in all cases. On average almost 68.4% decrease in memory was found. Experiments conducted using 18 attribute sets have shown that Dep(DDC) reduces the execution time almost by 63% as compared to positive region based approach Dep(P). Dep(DDC) consumes less memory than Dep(P). Results indicated that Dep(DDC) reduced the required runtime memory almost by 96% for eighteen datasets with different number of attributes and records. The reason behind is that Dep(DDC) does not require to calculate equivalence class structure as required by the first two steps of Dep(P).

In second step performance of algorithms using dependency classes based dependency calculation methods was analyzed. Again results justified the efficiency and effectiveness of the algorithms using dependency classes based methods. These results support the argument that the proposed IDC can be more effective to enhance the performance of algorithms based on calculation of dependency using positive region. Calculations show that Overall IDC based algorithms resulted in almost 65.93% decrease in execution time. The main reason behind the efficiency of IDC is that it avoids the calculation of the time consuming positive region calculation. The theory behind IDC is that it is the ratio of total number of attribute values that lead towards unique decision to total universe size. This is the point which helps reduce the execution time in case of IDC. Conventional rough set based approach on the other hand first creates equivalence class structure w.r.t. the decision attribute(s), then equivalence class structure using the attributes under consideration (on which dependency is to be measured) and finally the positive region is calculated. These steps consume too much time which makes positive region based dependency measure method totally unsuitable for other than smaller datasets. In our experiments (both in positive region based algorithms and those using IDC), we used intermediate data structures in their global scope. Maximum required memory was allocated at the start of algorithm and was utilized throughout to

avoid run time creation/deletion of memory (consequently to enhance performance). Results have shown that algorithms using IDC consumed less memory as compared to positive region based approach in all cases. Overall 68.4% decrease in runtime required memory was found in algorithms using IDC.

Similarly experiments showed that algorithms using DDC based method show a significant decrease in execution time. We have observed decrease of almost 95% for six datasets. This justifies our claim that algorithms using DDC method are more efficient as compared to those using positive region based approach. Algorithms using DDC have taken less memory as compared to positive region based approaches. Results have shown that DDC based approaches have shown almost 95% decrease in memory on average for six datasets. The reason behind is that DDC based approaches require only one matrix to calculate number of unique and non-unique classes. Positive region based approaches on the other hand require two matrices to calculate equivalence class structure

8.5 Future Work

The following has been suggested for future.

8.5.1 Dependency Classes For Unsupervised Learning

So far dependency classes were used only for supervised datasets. This is because dependency classes require class labels to predict the degree of dependency as new record is read. This happens in case of both incremental dependency classes and direct dependency classes. However, we may come along the situation where data is not labelled i.e. in case of unsupervised learning. So the application of dependency classes may be challenging in this case and a scenario still to be tested. As part of our future work we intend to apply dependency classes for unsupervised model and motivation comes from [88] where authors have presented unsupervised QuickReduct algorithm. This would be beneficial in many real world applications including clinical decision support systems, Vehicle identification and tracking and weather forecasting etc.
Figure-8.1shows the pseudo code of the algorithm.

USQR (C)

C, the set of all conditional features;

(1) $R \leftarrow \{\}$

(2) do

(3) $T \leftarrow R$

(4) $\forall x \in (C - R)$

(5) $\forall y \in C$

(6) $\gamma_{R \cup \{x\}}(y) = \dfrac{|\mathrm{POS}_{R \cup \{x\}}(y)|}{|U|}$

(7) if $\overline{\gamma_{R \cup \{x\}}(y), \forall y \in C} > \overline{\gamma_T(y), \forall y \in C}$

(8) $T \leftarrow R \bigcup \{x\}$

(9) $R \leftarrow T$

(10) until $\overline{\gamma_R(y), \forall y \in C} = \overline{\gamma_C(y), \forall y \in C}$

(11) return R

Figure-8.1: unsupervised QuickReduct taken from [88]

Authors have used positive region based dependency measure to calculate dependency of attributes, however, as in unsupervised learning we are not provided with data labels, so the dependency of attributes on each other is checked and the attributes that provide maximum increase in dependency of other attributes on them are selected. So instead of an explicitly available decision class, C – {x} attributes act as decision class for attribute {x}. However, the drawback with this approach is that using positive region based approach makes it impossible to apply for large datasets, so our intension is to use dependency classes where features could be selected without applying positive region. We have attempted to dry run incremental dependency classes to calculate dependency using dependency classes.

Table-8.2 taken from [88] shows the dataset used for experimentation.

Table-8.2: Sample IS taken from [88]

U	a	b	c	d
1	1	0	2	1
2	1	0	2	0
3	1	2	0	0
4	1	2	2	1
5	2	1	0	0
6	2	1	1	0
7	2	1	2	1

Dependency calculated by unsupervised QuickReduct algorithm for the datasets given in Table-8.3

Table-8.3: Dependency values calculated by [88]

y|x	{a}	{b}	{c}	{d}
A	1	1	0.1429	0
B	0.4286	1	0.1429	0
C	0	0.2857	1	0.4286
D	0	0	0.4286	1

Where each column specifies the degree of dependency of other attributes on a particular attribute e.g. third row specifies value of attribute {b} on {a}, fourth row specifies degree of dependency of {c} on {a} and fifth row specifies degree of dependency of {d} on {a}. We have dry run dependency classes for calculating dependency and results obtained when compared with those calculated by positive region based dependency measure were same. Following section shows how dependency was calculated in unsupervised mod for these attributes using dataset given in Table-8.3.

Here we need to calculate dependency of {a} on {a}, {b} on {a}, {c} on {a} and {d} on {a}. For description purpose we will only calculate dependency of {b} on {a} using IDC. So, in this case {b} will be considered as decision class and {a} will be considered as conditional attribute.

So if we read the first record, value of attribute {a} is "1", decision class is "0", since record appears for the first time, so as per rules of IDC, the applied IDC will be "Initial Positive Region" class. Since UDV value before reading this record was "0" (as before this we had no record read) so dependency will be:

$$\gamma(c, D) = \frac{UDV + 1}{|U'| + 1}$$

$$\gamma(a, b) = \frac{0 + 1}{|0| + 1}$$

$$\gamma(a, b) = \frac{1}{1}$$

$$\gamma(a, b) = 1$$

Value of UDV becomes "1" as we have only one unique record so far. After reading second record i.e. object "2", we see that this object has already appeared with same vale of condition and decision attributes, so applied dependency class will be "Positive Region" class. Dependency will be calculated as:

$$\gamma(c, D) = \frac{UDV + 1}{|U'| + 1}$$

$$\gamma(a, b) = \frac{1 + 1}{|1| + 1}$$

$$\gamma(a, b) = \frac{2}{2}$$

$$\gamma(a, b) = 1$$

Value of UDV will become "2" because so far we have two unique records. Similarly after reading all the records and applying relevant dependency classes,

$$\gamma(a, b) = 3/7$$

After calculating dependency of all attributes, we have found dependencies equal to that calculated by unsupervised QuickReduct.

Similar to incremental dependency classes, direct dependency classes were also applied to calculate dependency. If we consider total number of unique classes then by formula:

$$k = \gamma(C, D) = \frac{Total\ number\ of\ unique\ classes}{|U|}$$

$$k = \gamma(a, b) = \frac{3}{|7|}$$

$$k = \gamma(a, b) = \frac{3}{7}$$

However, if we consider number of non-unique classes then dependency can be calculated by formula:

$$k = \gamma(C, D) = 1 - \frac{Total\ number\ of\ non - unique\ classses}{|U|}$$

$$k = \gamma(C, D) = 1 - \frac{4}{|7|}$$

$$k = \gamma(C, D) = \frac{3}{7}$$

Similarly we calculated dependency of all attributes using direct dependency classes (DDC), and result was found to be equal to that calculated by unsupervised QuickReduct.

However, this was only experimented using the above given dataset and only accuracy was measured. Other two components of comparison framework i.e. percentage decrease in execution time and memory usage was not analyzed and will be worked on as part of future work.

8.5.2 Dependency Classes For Unsupervised Feature Selection Algorithms

There are algorithms for feature selection designed for unsupervised mode e.g. feature selection for detecting social behaviour in case of social media applications, documents classification and fraud detection in banking applications. As far as accuracy is concerned, we can use dependency classes as replacement for positive region based approaches but the computation time and memory usage factors still need to be tested. If dependency classes prove to be successful for unsupervised

models in terms of all components of comparison framework, it will be a good help for performing unsupervised feature selection in large datasets by enhancing the efficiency and effectiveness of the underlying algorithms. Figure-8.2 highlights the steps where dependency classes based dependency measure can be used instead of positive region based dependency calculation.

USQR (C)
C, the set of all conditional features;
(1) $R \leftarrow \{\}$
(2) do
(3) $T \leftarrow R$
(4) $\forall x \in (C - R)$
(5) $\forall y \in C$
(6) $\gamma_{R\cup\{x\}}(y) = \frac{|\mathrm{POS}_{R\cup\{x\}}(y)|}{|U|}$
(7) if $\overline{\gamma_{R\cup\{x\}}(y), \forall y \in C} > \overline{\gamma_T(y), \forall y \in C}$
(8) $T \leftarrow R \cup \{x\}$
(9) $R \leftarrow T$
(10) until $\overline{\gamma_R(y), \forall y \in C} = \overline{\gamma_C(y), \forall y \in C}$
(11) return R

Figure-8.2: Unsupervised QuickReduct with highlighted adaptations taken from [88]

8.5.3 Dependency Classes For Other Algorithms

So far dependency classes have been applied to feature selection algorithms only. However a number of other algorithms including prediction algorithms, decision making algorithms, rule extraction algorithms etc. also use positive region based rough set dependency measure. So, apparently we can conclude that dependency classes can also be applied in these algorithms. However their effectiveness still needs to be analyzed and can be a track for future to further proof benefits of dependency classes.

8.6 Final Word

The aim of this investigation was to investigate methods capable of achieving dimensionality reduction which are also computationally effective. The main objective of RST is to reduce data size. Traditional rough set based approaches use positive region based dependency measure for feature selection process. However, using positive region is computationally expensive approach that makes it inappropriate to use for large datasets. We have developed an alternate way to calculate dependency comprising of dependency classes. A dependency class is a heuristic which defines how the dependency measure changes as we scan new records during traversal of the dataset. On the basis of the heuristics used by dependency classes, two types of dependency classes were proposed i.e. Incremental Dependency Classes (IDC) and Direct Dependency Classes (DDC). Experimental results justified the efficiency and effectiveness of proposed solution. Future directions include dependency classes for unsupervised datasets, dependency classes for unsupervised feature selection and dependency classes for algorithms other than feature selection.

References

[1] R. Bellman, "Dynamic programming: Princeton Univ. press." (1957).

[2] R. Jensen and Q. Shen. “Computational intelligence and feature selection: rough and fuzzy approaches.” Vol. 8. John Wiley & Sons, 2008.

[3] J. Hua, D. T. Waibhav and E. R. Dougherty. "Performance of feature-selection methods in the classification of high-dimension data." Pattern Recognition 42.3 (2009): 409-424.

[4] E. Hancer et al. "A multi-objective artificial bee colony approach to feature selection using fuzzy mutual information". 2015 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2015.

[5] G. H. John, R. Kohavi and K. Pfleger. "Irrelevant features and the subset selection problem." Machine learning: proceedings of the eleventh international conference. 1994.

[6] Z. Pawlak. "Rough sets-theoretical aspect of reasoning about data." (1991): 29-29.

[7] M. Podsiadło and H. Rybiński. "Rough sets in economy and finance." Transactions on Rough Sets XVII. Springer Berlin Heidelberg, 2014. 109-173.

[8] V. Prasad, T. S.Rao and M. S. P. Babu. "Thyroid disease diagnosis via hybrid architecture composing rough data sets theory and machine learning algorithms." Soft Computing 20.3 (2016): 1179-1189.

[9] C. Xie, L. Yong-Jun and J. Y. Chang. "Medical image segmentation using rough set and local polynomial regression." Multimedia Tools and Applications 74.6 (2015): 1885-1914.

[10] G. A. Montazer, S. ArabYarmohammadi. “Detection of phishing attacks in Iranian e-banking using a fuzzy–rough hybrid system.” Applied Soft Computing 35 (2015): 482-492.

[11] M.P. Francisco, J.V. Berna-Martinez, A.F. Oliva and M.A.A. Ortega, Algorithm for the detection of outliers based on the theory of rough sets, Decision Support Systems 75 (2015): 63-75.

[12] UCI machine learning repository: archive.uci.edu.pk. Access date: 02-Jan-2015.

[13] J. Yan et al. "Effective and efficient dimensionality reduction for large-scale and streaming data preprocessing." IEEE transactions on Knowledge and Data Engineering 18.3 (2006): 320-333.

[14] X. Zeng and S. Luo. "Generalized locally linear embedding based on local reconstruction similarity." In: Fifth IEEE International Conference on Fuzzy Systems and Knowledge Discovery (2008): 305-309.
[15] L. K. Saul et al. "Spectral methods for dimensionality reduction." Semisupervised learning (2006): 293-308.
[16] R. Liu et al. "Semi-supervised learning by locally linear embedding in kernel space", In: Proceedings of 19th IEEE International Conference on Pattern Recognition, (2008): 1-4.
[17] S. Gerber, T. Tasdizen, and R Whitaker. "Robust non-linear dimensionality reduction using successive 1-dimensional Laplacian eigenmaps." In: Proceedings of the 24th ACM International conference on Machine learning, (2007): 281-288.
[18] P. Cunningham. "Dimension Reduction." University College Dublin, Technical Report, 2007.
[19] H. Liu, H. Motoda, Computational Methods of Feature Selection, Chapman & Hall/Crc Data Mining and Knowledge Discovery Series, 2007.
[20] H. Almuallim and T.G. Dietterich. Learning with many irrelevant features. In the 9th National Conference on Artificial Intelligence, MIT Press, (1991): 547–552.
[21] B. Raman and R. I. Thomas. "Instance-based filter for feature selection." Journal of Machine Learning Research 1.3 (2002): 1-23.
[22] J. R. Quinlan. "C4. 5: Programming for machine learning." Morgan Kauffmann (1993): 38.
[23] H. H. Inbarani, A. T. Azar and G. Jothi. "Supervised hybrid feature selection based on PSO and rough sets for medical diagnosis." Computer methods and programs in biomedicine 113.1 (2014) 175-185.
[24] K. Zuhtuogullari, N. Allahverdi and N. Arikan. "Genetic Algorithm and Rough Sets Based Hybrid Approach for Reduction of the Input Attributes in Medical Systems." International Journal of innovative computing and information control 9 (2013) 3015-3037.
[25] Q. Wenbin, W. Shu, B. Yang and Z. Changsheng, "An Incremental Algorithm to Feature Selection in Decision Systems with the Variation of Feature Set." Chinese Journal of Electronics, 24 (2015) 128-133.

[26] Y. Chen, Q. Zhu and H. Xu. "Finding rough set reducts with fish swarm algorithm." Knowledge-Based Systems 81 (2015) 22-29.

[27] H. H. Inbarani, M. Bagyamathi and A. T. Azar. “A novel hybrid feature selection method based on rough set and improved harmony search.” Neural Computing and Applications (2015) 1-22.

[28] Y. Jiang and Y. Yu. “Minimal attribute reduction with rough set based on compactness discernibility information tree.” Soft Computing (2015) 1-11.

[29] Y. Qian, Q. Wang, H. Cheng, J. Liang and C. Dang. “Fuzzy-rough feature selection accelerator.” Fuzzy Sets and Systems 258 (2015) 61-78.

[30] A. Tan, J. Li, Y. Lin and G. Lin. “Matrix-based set approximations and reductions in covering decision information systems.” International Journal of Approximate Reasoning 59 (2015) 68-80.

[31] E. A. Daoud. “An Efficient Algorithm for Finding a Fuzzy Rough Set Reduct Using an Improved Harmony Search.” International Journal of Modern Education and Computer Science 2.8 (2015) 16-23.

[32] I. Park and G. SeokChoi. “Rough set approach for clustering categorical data using information-theoretic dependency measure.” Information Systems 48 (2015) 289-295.

[33] L. V. D. Maaten, E.O. Postma and H. J. V. D. Herik, “Technical report on Dimensionality Reduction: A Comparative Review.” http://ticc.uvt.nl/~lvdrmaaten/Laurens_van_der_Maaten/Publications.html. Access Date 15-Jan-2015.

[34] Y. Shi, and R. Eberhart. “A modified particle swarm optimizer.” In: proceedings of IEEE International Conference on Evolutionary Computation (1998): 69-73.

[35] C. Bai, D. Dhavale and J. Sarkis. "Complex investment decisions using rough set and fuzzy c-means: An example of investment in green supply chains." European Journal of Operational Research 248.2 (2016): 507-521.

[36] J. D. Cabedo and J. M. Tirado. “Rough Sets And Discriminant Analysis Techniques For Business Default Forecasting.” Fuzzy Economic Review 20.1 (2015): 3-37.

[37] S. Kusi-Sarpong et al. "Green supply chain practices evaluation in the mining industry using a joint rough sets and fuzzy TOPSIS methodology." Resources Policy 46 (2015): 86-100.

[38] S. Mishra et al. "Importance of Location Classification using Rough Set Approach for the Development of Business Establishment." International Journal of Computer Applications 119.24 (2015).

[39] B. S. Panda, S. S. Gantayat and A. Misra. "Rough set rule-based technique for the retrieval of missing data in malaria diseases diagnosis." Computational Intelligence in Medical Informatics. Springer Singapore (2015): 59-71.
[40] S. Pramanik and K. Mondal. "Cosine similarity measure of rough neutrosophic sets and its application in medical diagnosis." Global Journal of Advanced Research 2.1 (2015): 212-220.
[41] Prasad, V., T. Srinivasa Rao, and M. Surendra Prasad Babu. "Thyroid disease diagnosis via hybrid architecture composing rough data sets theory and machine learning algorithms." Soft Computing (2015): 1-11.
[42] V. Prasad, T. S. Rao and M. S. P. Babu. "Thyroid disease diagnosis via hybrid architecture composing rough data sets theory and machine learning algorithms." Soft Computing (2015): 1-11.
[43] K. K. Ghanyet al. "A rough set-based reasoner for medical diagnosis." In: Proceedings of IEEE International Conference on Green Computing and Internet of Things (2015): 429-434.
[44] C. Xie. L. Yong-Jun and J. Chang. "Medical image segmentation using rough set and local polynomial regression." Multimedia Tools and Applications 74.6 (2015): 1885-1914.
[45] A. Salam, A. Kalam and A. V. Deorankar. "Assessment on Brain Tumor Detection using Rough Set Theory." Intemational Journal of Advance Research in Computer Science and Management Studies, 2327782.3 (2015).
[46] R. Vashist and A. Vashishtha. "An Investigation into Accuracy of CAMEL Model of Banking Supervision Using Rough Sets." Computational Intelligence Applications in Modeling and Control. Springer International Publishing (2015): 1-25.
[47] L. Chen and Chih-Tsung Tsai. "Data mining framework based on rough set theory to improve location selection decisions: A case study of a restaurant chain." Tourism Management 53 (2016): 197-206.
[48] A. Janusz et al. "Rough Set Tools for Practical Data Exploration", Rough Sets and Knowledge Technology. Springer International Publishing (2015): 77-86.
[49] H. Chen et al. "A decision-theoretic rough set approach for dynamic data mining." Fuzzy Systems, IEEE Transactions on 23.6 (2015): 1958-1970.
[50] J. W. Grzymala-Busse. "A Rough Set Approach to Incomplete Data." Rough Sets and Knowledge Technology. Springer International Publishing (2015): 3-14.

[51] R.L .Villars, C.W. Olofson and M. Eastwood. “Big data: what it is and why you should care.” (http://www.tracemyflows.com/uploads/big_data/idc_amd_big_data_whitepaper.pdf), access date: 14-Feb-2016.

[52] S. Kaisler, F. Armour, J. A. Espinosa and W. Money. “Big Data: Issues and Challenges Moving Forward.” In: Proceedings of the 46th IEEE Hawaii International Conference on System Sciences (2013): 995–1004.

[53] N. Jothi , N. A. Rashid, W. Husain. “Data Mining in Healthcare – A Review.” Procedia. Computer Science 72 (2015): 306 – 313.

[54] Y. Han et al. "Semisupervised feature selection via spline regression for video semantic recognition." In: IEEE Transactions on Neural Networks and Learning Systems 26.2 (2015): 252-264.

[55] C. Boutsidis et al. "Randomized dimensionality reduction for-means clustering." In: IEEE Transactions on Information Theory 61.2 (2015): 1045-1062.

[56] M. B. Cohen et al. "Dimensionality reduction for k-means clustering and low rank approximation." In: Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing (2015):163-172.

[57] J. Bourgain, D. Sjoerd and Je. Nelson. "Toward a unified theory of sparse dimensionality reduction in euclidean space." Geometric and Functional Analysis 25.4 (2015): 1009-1088.

[58] F. Radenović. J. Hervé and O. Chum. "Multiple measurements and joint dimensionality reduction for large scale image search with short vectors." In: Proceedings of the 5th ACM International Conference on Multimedia Retrieval (2015): 587-590.

[59] A. T. Azar and A. E. Hassanien. "Dimensionality reduction of medical big data using neural-fuzzy classifier." Soft computing 19.4 (2015): 1115-1127.

[60] E. J. Candes, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? Journal of the ACM, 58(3):11:1–11:37, 2011.

[61] Kao, Yi-Hao, and Benjamin Van Roy. "Learning a factor model via regularized PCA." Machine learning 91.3 (2013): 279-303.

[62] K. R. Varshney and A. S. Willsky. "Linear dimensionality reduction for margin-based classification: high-dimensional data and sensor networks." In: IEEE Transactions on Signal Processing 59.6 (2011): 2496-2512.

[63] A. Gisbrecht, A. Schulz and B. Hammer. "Parametric nonlinear dimensionality reduction using kernel t-SNE." Neurocomputing 147 (2015): 71-82.

[64] L. Gottlieb and R. Krauthgamer. "A nonlinear approach to dimension reduction." Discrete & Computational Geometry 54.2 (2015): 291-315.

[65] A. Gisbrecht and B Hammer. "Data visualization by nonlinear dimensionality reduction." Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 5.2 (2015): 51-73.

[66] B. Abolhasanzadeh. "Nonlinear dimensionality reduction for intrusion detection using auto-encoder bottleneck features." In: Proceedings of 7th IEEE Conference on Information and Knowledge Technology (2015):1-5.

[67] X. Wei, M. Kleinsteuber and H. Shen. "Invertible Nonlinear Dimensionality Reduction via Joint Dictionary Learning." Latent Variable Analysis and Signal Separation. Springer International Publishing (2015): 279-286.

[68] M. Alessio and C. V. Cannistraci. "Nonlinear Dimensionality Reduction by Minimum Curvilinearity for Unsupervised Discovery of Patterns in Multidimensional Proteomic Data." 2-D PAGE Map Analysis: Methods and Protocols (2016): 289-298.

[69] b. Tang, S. Kay and H. Haibo. "Toward optimal feature selection in naive Bayes for text categorization." In: IEEE Transactions on Knowledge and Data Engineering 28.9 (2016): 2508-2521.

[70] F. Jiang, S. Yuefei and Lin Zhou. "A relative decision entropy-based feature selection approach." Pattern Recognition 48.7 (2015): 2151-2163.

[71] D. Singh et al. "Feature Selection Using Rough Set For Improving the Performance of the Supervised Learner." International Journal of Advanced Science and Technology 87 (2016): 1-8.

[72] J. Xu et al. "L1 graph based on sparse coding for feature selection." International Symposium on Neural Networks. Springer Berlin Heidelberg (2013): 594–601.

[73] N. Hamdi, K. Auhmani and M. M. Hassani. "Quantum Clustering-Based Feature Subset Selection for Mammographic Image Classification." International Journal of Computer Science & Information Technology 7.2 (2015): 127-133

[74] G. Roffo, S. Melzi and M. Cristani. "Infinite Feature Selection." In: Proceedings of the IEEE International Conference on Computer Vision (2015): 4202-4210.

[75] X. He, D. Cai and P. Niyogi. “Laplacian score for feature selection.” Advances in Neural Information Processing Systems 18 (2005): 21-26.

[76] M. Devaney and R. Ashwin. "Efficient feature selection in conceptual clustering." In: Proceedings of the Fourteenth International Conference on Machine Learning (1997): 92-97.

[77] M. Gluck. "Information, uncertainty and the utility of categories." In: Proceedings of the Seventh Annual Conf. on Cognitive Science Society (1985): 283-287.

[78] J. Yang, X. Hua and J. Peifa. "Effective search for genetic-based machine learning systems via estimation of distribution algorithms and embedded feature reduction techniques." Neurocomputing 113 (2013): 105-121.

[79] M. B. Imani, M. R. Keyvanpour and R. Azmi. "A novel embedded feature selection method: a comparative study in the application of text categorization." Applied Artificial Intelligence 27.5 (2013): 408-427.

[80] M. Viola et al. "A generalized eigenvalues classifier with embedded feature selection." Optimization Letters (2015): 1-13.

[81] S. Maldonado et al. "Feature selection for support vector machines via mixed integer linear programming." Information sciences 279 (2014): 163-175.

[82] Z. Xiao et al. "ESFS: A new embedded feature selection method based on SFS." Rapports de recherché (2008).

[83] L. Du and S. Yi-Dong. "Unsupervised Feature Selection with Adaptive Structure Learning." In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2015): 209-218.

[84] J. Li et al. "Unsupervised Streaming Feature Selection in Social Media." In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (2015): 1041-1050.

[85] D. A. A. G. Singh, S. A. A. Balamurugan and E. J. Leavline. "An unsupervised feature selection algorithm with feature ranking for maximizing performance of the classifiers." International Journal of Automation and Computing 12.5 (2015): 511-517.

[86] P. Zhu et al. "Unsupervised feature selection by regularized self-representation." Pattern Recognition 48.2 (2015): 438-446.

[87] N. Zhou et al. "Global and local structure preserving sparse subspace learning: an iterative approach to unsupervised feature selection." Pattern Recognition 53 (2016): 87-101.

[88] C. Velayutham and K. Thangavel. "Unsupervised quick reduct algorithm using rough set theory." Journal of Electronic Science and Technology 9.3 (2011): 193-201.