Comparison of Classification Ability of Hyperball Algorithms to Neural Network and K-Nearest Neighbour Algorithms

Tanaby Zibamanzar Mofrad

Department Of Computer Science

Submitted in partial fulfillment
of the requirements for the degree of

Master of Science

Faculty of Mathematics and Science, Brock University
St. Catharines, Ontario

To my lovely wife, Tina and my respected parents, Pari and Farhad.

ABSTRACT

The main focus of this thesis is to evaluate and compare Hyperball learning algorithm (HBL) to other learning algorithms. In this work HBL is compared to feed forward artificial neural networks using back propagation learning, K-nearest neighbor and ID3 algorithms.

In order to evaluate the similarity of these algorithms, we carried out three experiments using nine benchmark data sets from UCI machine learning repository. The first experiment compares HBL to other algorithms when sample size of dataset is changing. The second experiment compares HBL to other algorithms when dimensionality of data changes. The last experiment compares HBL to other algorithms according to the level of agreement to data target values.

Our observations in general showed, considering classification accuracy as a measure, HBL is performing as good as most ANn variants. Additionally, we also deduced that HBL's classification accuracy outperforms ID3's and K-nearest neighbour's for the selected data sets.

List of Tables

List of Figures

Contents

Chapter 1

Introduction

This thesis deals with a comparative study of Hyperball (HBL) algorithm as a new classification algorithm to back propagation learning [1] in feedforward artificial neural network as well as K-Nearest neighbour [2] and decision tree [3]. In this thesis, accuracy of three variants of HBL algorithm is compared to four variants of back propagation neural networks.

Artificial neural network (ANN) is a computational model which tries to simulate structure or functional behaviour of biological neural networks. It consists of a number of nodes which are connected to each other using weighted connections. Different arrangements of nodes together construct different layers of ANNs. In multi layer ANNs, at least three layers of nodes are formed: An input layer which receives input to the network, hidden layers which can assist in the classification of non-linearly separable patterns; and an output layer which is used for actual output of the neural network.

Feedforward neural networks are a type of ANNs where the weight connections between the nodes do not form a directed cycle. In this type of ANN, information only flows in one direction, which is from the input layer to the output layer. By using back propagation learning,

multi layer neural networks can be trained to learn the input patterns and achieve a form of generalization by which they can also classify unseen similar patterns. In its simplest form, BP uses gradient descent techniques. Back propagation learning calculates the error of the output nodes and propagates the error backwards to tune the weights of the neural network using local gradient of output nodes.

There are many variants to gradient descent back propagation learning. In this thesis, Levenberg-Marquardt (LM) [4], Resilient Back Propagation (RProp) [5], Quasi-Newton (BFG)[6] and standard Gradient Descent with Momentum (GDM) [1] are used as four variants of back propagation learning. Levenberg-Marquardt and Quasi-Newton variants make use of second order information about the error surface which is represented by the local Hessian matrix. Quasi-Newton tries to approximate the inverse of Hessian matrix over a number of steps instead of computing Hessian which is computationally expensive. Levenberg-Marquardt is specifically designed to locate the minimum of sum-of-squares error function which is used to calculate the error at the output layer of neural networks. Resilient back propagation tries to use the sign of partial derivatives in gradient descent to indicate the direction of weight update.

One of the main works of this thesis is to evaluate HBL algorithm in comparison to neural networks. HBL algorithms are recently introduced in [7] and can be used for both supervised and unsupervised learning. However, in this thesis HBL is used for supervised learning.

HBL algorithms work similarly to the category of Instance Based Learning (IBL) algorithms [2]. These algorithms try to classify patterns based on computing a similarity-measure (distance) to the patterns they have learned before. Classification of a new pattern is done by the contribution of a number of more similar (less distant) patterns to the new pattern. When a pattern is introduced to HBL, it centers a ball around

that pattern. This ball is a secure margin for the pattern which can contain other very-similar patterns. Each ball is labeled to the class to which the pattern belongs. So all balls of the same class are labeled uniquely which construct a category. The number of categories is equal to the number of classes associated with the dataset in supervised learning. Radii of the balls in each category is decided and altered over a number of steps. Balls which belong to same category can overlap whereas balls which belong to different categories are shrunk to be mutually exclusive.

HBL in testing mode examines the patterns in the balls of each category looking for a number of more similar patterns to the testing pattern. It then uses the information from this group of patterns to classify the testing pattern. In this thesis, three variants of HBL algorithm are introduced.

Like any other classification algorithm, HBL also has both advantages and disadvantages. Being a category of Instance Based Learning algorithms, HBL can construct a different approximation for each distinct testing pattern which is to be classified. This can be advantageous when the target function is very complex because HBL can describe it as a collection of local approximations [2]. One disadvantage of HBL is that the cost of classifying testing patterns can be higher when the number of instances increase because the number of distance calculations increases as well. Another disadvantage of HBL is that all of the attributes associated with a pattern are considered by the algorithm for classification and they are all given equal importance whereas in practice not all attributes are of equal importance to describe data.

In addition to further evaluate the performance of HBL, K-Nearest Neighbour (KNN) [2] and decision trees ID3 [3] are also included in the comparison. Just like HBL algorithm, KNN algorithm also belongs to a category of Instance Based Learning. ID3 algorithm constructs a decision tree by calculating information gain of each of the unused attributes and

selects the attribute with maximum information gain as a node. This process is continued until all of the attributes associated with data are used.

In order to compare HBL to ANN, KNN and ID3 algorithms, 9 benchmark datasets are selected from the UCI machine learning repository and generalization accuracies of each algorithm over these datasets are used as a measure of comparison. The selected datasets are: Glass, Iris, Pima Indian Diabetes, Connectionist, Madelon, Wine, Zoo, Breast Cancer, Musk Version two, and Parkinson dataset.

This thesis focuses on a few different criteria which can influence the accuracy of a classifier and conducts an experiment for each to compare HBL to other algorithms. These criteria are sample size and number of attributes associated with each pattern (data dimensionality). It also uses another criterion as the level of agreement between a classifier and the actual data output to compare classifiers.

The first experiment tries to compare HBL to other algorithms when the sample size is increasing. For this experiment, we have divided the datasets into two groups and have conducted the experiment for each group of dataset when each time sample size is increasing by 25%.

The second experiment compares HBL to other algorithms when the dimensionality of data changes in each dataset. Principal Component Analysis (PCA) [8] is employed to reduce the dimensionality and each time increase number of selected principal components by 25%. For this experiment, datasets with equal number of classes are grouped together and their accuracy results are averaged over their group. The third experiment compares HBL to various other algorithms in order to determine whether if in general HBL algorithm has a better level of agreement to actual target values. In general for this experiment, Kappa statistical test [9] is utilized.

The remainder of the thesis is organized as follows: Chapter 2 will give an introduction to artificial neural networks. Back propagation learning algorithm is then explained in details in this chapter. This chapter also explains the other variants of back propagation algorithm which are used in this thesis. In chapter 3, KNN and ID3 algorithms are explained in detail. It also describes the different learning categories each of these classifiers belong to. Next in chapter 4, HBL and its different variants are introduced. Chapter 5 performs a literature review on different KNN variants to see if they are similar to HBL algorithm. Chapter 6 describes the datasets we are using along with experimental setups. Selected datasets are briefly described in a tabular form and for each classifier, experimental details and parameters' initial values are explained. Actual experiments and results are shown and discussed in chapter 7. Lastly chapter 8 describes the conclusions and proposes the future works which can be performed and established in addition to this thesis.

Thesis Goals

In this work we have tried to achieve following goals:

1. Compare performance (classification accuracy) of HBL algorithm to ANNs using back propagation, KNN algorithm, and ID3 algorithms for a number of selected data sets and when the sample size increases. Use student's paired t-test and effect size[10] (Shows the statistical strength of a difference) to determine the significance and importance of each comparison.

2. Compare performance (classification accuracy) of HBL to other classifiers in terms of classification accuracy on a set of data sets when

dimensionality of data changes. Use student paired t-test and effect size to determine the significance and importance of difference.

3. Using Cohens Kappa test of statistics [11] determine the level of agreement between each classifier's actual output and target output. Apply a Z-Statistical test [12] to determine the significance of obtained values. Then using these values, compare HBL to these other algorithms.

Chapter 2

Artificial Neural Networks: Back Propagation Learning

2.1 Introduction to Artificial Neural Networks (ANN)

An artificial neural network [13] (ANN) is a model which is used to simulate both the structure and functional behavior of a biological neural network. It is comprised of a group of artificial neurons which are interconnected. Artificial neurons are connected to each other by connection lines which are called "weight connections". Artificial neurons and weight connections together form the architecture of a neural network.

All neural networks should be trained prior they can perform a task with some sort of training rule. Using training rule weight connections will be updated on the basis of data and feedback. If neural networks are trained properly then they can show capability for generalization beyond the training data. This means they can approximately produce correct results for the cases which are not included in the training data but belong to same data category.

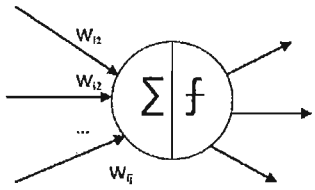

Figure 2.1: A simple Neuron

2.1.1 A simple Artificial Neuron

An artificial neuron (is often called as node) receives input from other nodes or from external environment. Each input to a node is comprised of a weight value w. In each node, all of the weighted inputs are summed together and then a function f calculates the weighted sum of its inputs as follows:

$$y_i = f(\sum_{i=1} w_{ij} y_i) \tag{2.1}$$

$$y_i = f(net_i) \tag{2.2}$$

In Figure 2.1 , the weighted sum $\sum_j w_{ij} * y_j$ is called the net input to node i, net_i where y is an output from a node. Note that w_{ij} refers to the weight from node j to node i. The function f is the node's activation function. In the simplest case, f is the identity function, and the node's output is just its net input. This is called a linear node (because its output is equal to its net value), however it is nonlinearity which gives ANNs capability to classify nonlinearly separable data. ANNs can have different architectures and models. Multi layer ANN models have been used in

this thesis. These models are explained in the following section.

2.1.2 Multi layer ANNs

Multi layer neural networks have at least three different layers of nodes . The first layer is the input layer, consisting of nodes which receive input from the outside. The last layer is called the output layer, which is the neural network's output and transfers the response of the whole neural network to the outside environment. In between, there can be at least one hidden layer of hidden nodes which process the received input from the preceding layers. Each neuron has an activation (squashing) function associated with it. If a network consists of linear activation functions, then it can learn to classify patterns which can be just linearly separable [14]. Linearly separable patterns are patterns by which any two different classes of patterns can be separated by a straight line (plane, or hyperplane depending on dimensionality of data patterns) [14]. If the activation functions associated with hidden layers are nonlinear functions, then the network can also learn to classify the non linearly separable patterns. Activation functions for hidden units can introduce nonlinearity to the network. Without nonlinearity, hidden units would not make neural networks more powerful than plain perceptrons (which do not have any hidden units, only input and output units). The reason is that a linear function of linear functions is again a linear function (which is the case in perceptron networks).

Shown in figure 2.2 is a network composed of three layers. Input units receive the input values. In each hidden unit, the sum $net_j = \sum_j w_{ji} * y_j$ and $f(net_j)$ is calculated. The same process is repeated in the output layer. There are different algorithms and training models associated with multi layer neural networks. Back Propagation (BP) algorithm is a well known

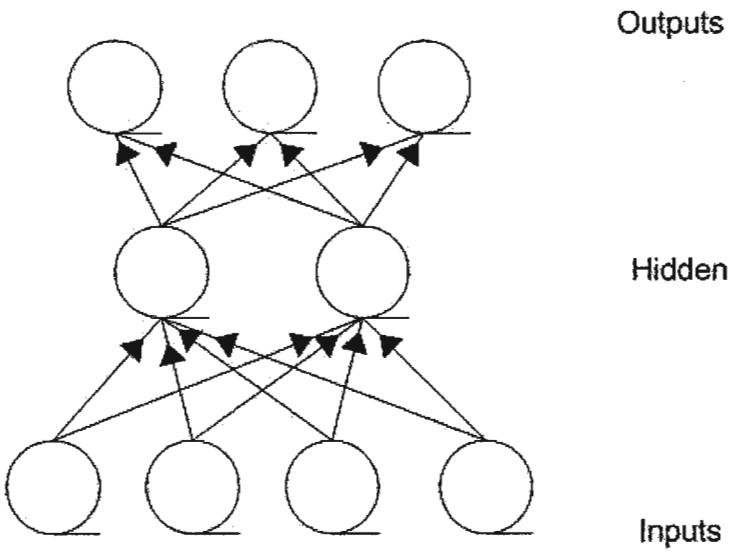

Figure 2.2: A simple multi layer Network

algorithm which can be used in multi layer neural networks to adjust the weights and train the network [15].

2.1.3 Back Propagation Algorithm

If we consider a multi layer neural network, the final layer of weights can be considered as a perceptron with inputs given by the outputs of the last hidden layer. These weights can be chosen and tuned using the perceptron learning rule [16] which is a rule for tuning and choosing the weights and bias in perceptron learning. However, the weights in earlier layers of our network cannot be determined by these rules. Another approach can be to consider each layer individually as a single layer perceptron but then we can not associate target values with those layers anymore. Indeed, such networks cannot be trained using perceptron procedures. The solution to the above problem is relatively simple. If we consider a net-

work with differentiable activation functions then the activation of output nodes become differentiable functions of both the input variables and the weights and biases [1]. If we define the error function as sum-of-squares error (where the error is difference between the actual network output and the target value), then this error itself is a function of the weights which is differentiable as well [1]. We can then calculate the derivatives of error with respect to weights, and these derivatives can be used to calculate the required adjustments in the weights values. The algorithm which evaluates the derivatives of weight functions is called back propagation algorithm [15] since it tries to propagate the error backwards from the output layer to the previous layers. With respect to partial derivative of the error over corresponding weight, it then finds the new weight value. Back propagation pseudo code is presented below:

Algorithm 1 Back Propagation

1: Begin
2: Initialize the weights to some random values
3: **repeat**
4: **for** each input pattern n in the training set **do**
5: compute $y(n)$ as the actual output of the network
6: Calculate error $(y(n) - d(n))$ at the output units
7: Update all weights from hidden layer to output layer using:
8: $\Delta w_{ji} = \eta e_j(n) f'(net_j(n)) y_i(n)$
9: Update all weights from input layer to hidden layer using:
10: $\Delta w_{ji} = \eta f'_j(net_j(n)). \sum_k \delta_k(n) w_{kj}(n) y_i(n)$
11: **end for**
12: **until** all examples classified correctly or stopping criterion met
13: **return** network

As demonstrated above, back propagation algorithm has two phases: The first phase involves a forward propagation of inputs as explained in equations 2.1 and 2.2 (line 5 in above algorithm). In the second phase, error e can be calculated for every output node, and the sum of squares error,

E, can be calculated for the network. Then, derivations of E over weights are calculated to find the Δw_{ji} which is the required adjustment for the weight w_{ji}. In other words, after input is fed to the network, error is calculated at the output but error is then propagated backwards, so for each existing weight, Δw_{ji} is calculated. The update Δw_{ji} is $\eta e_j(n) f'(net_j(n)) y_i(n)$ and for other hidden-output weights and $\eta f'_j(net_j(n)).\sum_k \delta_k(n) w_{kj}(n) y_i(n)$ (lines 6 – 10 in above algorithm). In order to minimize the error function, an iterative procedure is needed for most training algorithms which adjusts the weights accordingly in each iteration. The derivation of back propagation learning algorithm is been discussed in details in appendix A.

Vanilla back propagation usually works for simpler models, however it might take a long time to converge if error space becomes more complex. The reason for this is that the places which small step sizes are needed can be problematic for the whole convergence. For example when descending a steep place, small step sizes are needed to be taken. On the other hand when descending in a gently sloping parts longer step sizes should be taken [17]. A possible solution to the problem is by choosing a step that is some constant times the negative gradient rather than a step of constant length in direction of the negative gradient [4]. This is equivalent to moving slowly in shallow regions and moving quickly in steep regions. Another issue is that the error surface curvature can be different in all directions and not the same. This will cause different components of the gradient in different directions to have different values which can slow down the algorithm [17].

ANN Variants

Error back propagation works by moving towards the negative direction of local gradient of error with respect to weights to find the minimum error and new weight values. One of the well known algorithms to find minimum of an error function is Newton's method [18]. Newton's methods can be used to find approximations to roots of a quadratic equation. Unlike gradient descent, Newton's methods point directly to the minimum of a function. The problem with Newton's methods is that they have to calculate Hessian matrix (matrix of second derivatives or error with respect to weights) to be able to proceed in the direction of descent. However, calculating Hessian matrix numerically needs a lot of computation and will be very costly. Quasi-Newton overcomes this drawback by using the curvature information from f and ∇f to approximate the Hessian matrix. The update formula for this algorithm is given in below:

$$G(n+1) = G(n) + \frac{PP^T}{P^T v} - \frac{(G(n)v)v^T G(n)}{v^T G(n) v} + (v^T G(n) v)uu^T \tag{2.3}$$

p, v, and u are vectors which are defined as:

$$p = w(n+1) - w(n) \tag{2.4}$$

$$v = g(n+1) - g(n) \tag{2.5}$$

$$u = \frac{p}{p^T v} - \frac{G(n)v}{v^T G(n) v} \tag{2.6}$$

Levenberg-Marquardt [17] is another variant of ANN back propagation which is designed to minimize the sum-of-squares cost function without computing Hessian matrix. This algorithm instead uses Jacobian matrix of partial derivatives of error with respect to connection weights for every training pattern. M. Poulton et. al in [19] state that Levenberg-Marquardt algorithm will look to minimize error only in a small search area where the linear approximation is possible. Levenberg-Marquardt benefits from a

step size factor whose value can greatly shift algorithm behavior towards Newton's method or gradient descent. If step size is chosen to be a large value, then algorithm will behave like Newton's method. On the other hand a small step size value will result in an algorithm equivalent to gradient descent. The update formula of this algorithm is given as below:

$$w(n+1) = w(n) - (Z^T Z + \lambda I)^{-1} Z^T E(w(n)) \tag{2.7}$$

Where

$$E(w(n+1)) = E(w(n)) + Z(w(n+1) - w(n)) \tag{2.8}$$

$$Z = \frac{\partial E}{\partial w} \tag{2.9}$$

Rprop or Resilient back propagation method [5] is another variant which can be used in supervised batch learning in multi layer perceptron networks. Rprop uses the sign of partial derivatives to indicate the direction of weight update so it eliminates the problem of calculating partial derivatives in weight steps in gradient descent. For each weight, if signs that a partial derivative of the total error function has changed when compared to last iteration, then the update value for that weight is multiplied by a coefficient $\eta^- < 1$. If there is no change in the sign, then the update value is multiplied by $\eta^+ > 1$. Finally, each weight is updated by its update value in the opposite direction of its partial derivative.

Rprop is a reliable variant when the size of a network is too big and dramatically slows down Levenberg Marquardt and Quasi-Newton algorithms [1].

Chapter 3

K-Nearest Neighbour, Decision Tree Method

3.1 K-Nearest Neighbour Algorithm (KNN)

In order to explain the KNN algorithm [20], let's assume that the data is divided into two main categories: training set and testing set. Let's choose a pattern from testing set, $P_{test}(a_1, a_2, a_3, ...a_n)$, which consists of n attributes (a). The idea in KNN algorithm is to dynamically select k patterns in training set (P_{train}) which are "more similar" to the pattern to be classified in testing set (P in above) [20]. Similarity can be identified by calculating the differences (distances) to each P_{train} in the training set and then choosing the k patterns which have more similarity (e.g smaller differences) to P_{test}. Then, KNN will use the information from k selected patterns to classify P_{test}. The problem is to find class_Index using $Class\_Index = f(a_1, a_2, ...a_p)$ equation. If function f was known then class_Index could be simply calculated. But f is not known so a reasonable idea will be to look for observations in our training data that are near it [20].

As we mentioned above, KNN uses the k closer pattern's information to classify the P_{test}. This is achieved by calculating distance and finding k

examples that their distance is closest to P_{test}. In classification problems, a majority voting scheme is employed for KNN's prediction [20].

The choice of k is considered to be an important factor in this algorithm. k is considered to be a smoothing parameter [2]. The reason is a small value of k in any given problem, will lead to large variances in predictions. If we choose a large value for k, then it could cause a large model bias [20]. So k should be chosen in such a way that it is large enough to minimize misclassification probability. On the other hand, it should be small with respect to number of instances so that k nearest points are close enough to the query point [20]. Thus, the optimal value for k should be chosen and selected. By using a cross validation technique, an estimate of k in KNN algorithm can be calculated [20, 1].

In order to make predictions with KNN, a metric needs to be defined for measuring the distance between the query point and cases from the training sample. In this thesis, Euclidean metric space is used for the K-nearest neighbor algorithm. The KNN's pseudo code for 10-fold cross validation is presented below:

Algorithm 2 K-Nearest Neighbour Algorithm using Cross Validation

1: Divide the dataset using 10 fold cross validation:
2: each time use one fold for testing and remaining 9 folds for training
3: **for** each test sample t do the following: **do**
4: determine the closest k training samples based on calculated distance.
5: determine w the most frequent class label among the available C classes.
6: Update corresponding confusion matrix.
7: **end for**
8: Make decisions based on confusion matrix (accuracy).

As we can see above, 10-fold cross validation is applied to partition data into 10 folds randomly. Each time, 9 folds are used for training

and one fold for testing. w, the most frequent class among C existing classes is chosen based on distance of each query point (test pattern) to K nearest stored patterns, and correspondingly, confusion matrix is been updated. Confusion matrix [21], is used to evaluate the performance of an algorithm using the data which is stored in the matrix.

There are a few shortcomings associated with KNN algorithm. First, the time to find the nearest neighbors in a large training set can be costly. Second, the number of observations required in the training data set to qualify as large, increases exponentially with the number of dimensions n [2].

Another disadvantage is when dimensionality of data increases finding nearest neighbor also will more cumbersome as number of calculations also increases. KNN variants will be later on explained in this work.

3.2 Decision Tree (Dtree) learning

Decision Trees [22] can be used for classification, prediction and functional approximation. In contrast to neural networks, decision trees represent rules. These rules can be as simple as a set of if-then statements. In order to classify instances, decision trees sort instances down the tree from root to some leaf node. Each node in a decision tree corresponds to a test of some attribute of instances and each branch descending from that node corresponds to a possible value for that attribute. An instance is classified by starting at the root node of the tree, testing the attribute and moving down the branch corresponding to the attribute in a given example. The same process is repeated for the subtrees rooted at that node. The induction of decision trees requires to come up with a classification rule which can use the attribute values of a pattern and determine the target class

which to the pattern belongs. This classification rule is expressed as a decision tree. In decision trees, leaves' nodes are classes (target values) associated with patterns. Other nodes in the tree are attribute based tests and branches are values which are associated with each attribute. The essence of decision tree induction is to determine whether decision trees can generalize to unseen patterns beyond the training set. It means the algorithm tries to construct a decision tree which can classify patterns from the testing set given that training set and testing set are mutually exclusive.

There can be more than one correct decision tree for a particular training set. In this case, smaller and less complex trees are preferred over more complex ones. Quinlan in [22] argues that the greater the complexity of a tree, the more the tree is likely to be an explanation of the training set. In Figure 3.1 a sample decision tree is shown for Iris data set. As it is shown in this figure, each node corresponds to a test of attribute and each branch descending from that node corresponds to a possible value for that attribute. Finally, each leaf node represents a classification decision in this decision tree.

3.2.1 ID3

A solution to the simpler-the-better (Occam's Razor) approach above is to construct all possible decision trees and then select the simplest one. When the datasets begin to get very large, it is not practical to construct the decision tree based on all samples of training set. In this case, a window as a subset of the training set sample size is selected and used to construct the decision tree. If the constructed decision tree cannot classify some of the patterns from training set, then those patterns are added to the window set. ID3 benefits from a few concepts from information

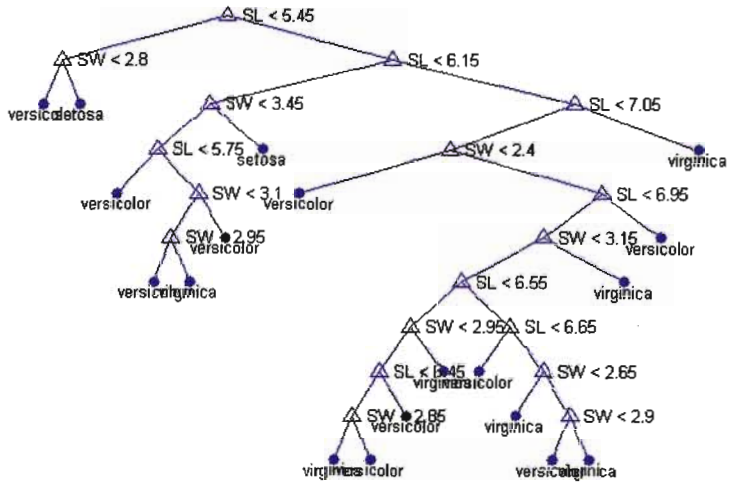

Figure 3.1: Decision Tree Classification

theory. Let us call training set as "S," then Entropy of S is defined in [23] as follows: " Entropy is a measure of impurity in the collection of training set".

Informally, entropy can be provide some information about data, in the sense that the higher the entropy is, the more information is required to completely describe that data. Entropy's formula is given in [2] as :

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \, log_2 \, p_i \tag{3.1}$$

p_i is the proportion of instances in training set that have *ith* value of the target set. Information gain $Gain(S, A)$ of an attribute A is calculated as follows:

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{Card(S_v)}{Card(S)} Entropy(S_v) \tag{3.2}$$

Values(A) is set of all possible values for attribute A and S_v is subset of S for which attribute A has value v. In decision trees, the first node (root) should be selected among different available attributes based on the score given by the information gain.

In order to construct the tree, the information gain for all attributes is calculated and the attribute with the highest information gain is selected. When we want to descend the tree, attributes should be selected in such a way that they reduce entropy, and attributes which have the most entropy reduction are most suitable to be chosen [22]. In the same manner information gain is also defined as the expected entropy reduction imposed by a specific attribute. ID3 algorithm for a two class problem, (0,1), is presented below exactly as it is given in [2].

Algorithm 3 corresponds to a two class classification of data. The first 7 lines of this algorithm correspond to conditions which can lead to the condition in which tree is single node tree. In lines 9 – 18 algorithm is trying to select best attribute with highest information gain and then grown the tree by adding branches based on possible values of the selected attribute. The algorithm calls itself recursively until the complete tree is constructed.

As explained earlier, information gain is a measure used by ID3 to select the best attribute in each step in expanding the tree. Stopping conditions can occur either when every attribute has already been included along a path throughout the tree, or when all of the training examples associated with a particular leaf node have the same target attribute value.

Algorithm 3 ID3 Algorithm

Examples are the training examples. Target_ attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given example.

```
1: Create a Root node for the tree.
2: if all Examples are positive then
3:    single-node tree Root, with label=plus.
4: else if all Examples are negative then
5:    the single-node tree Root, with label=minus
6: else if attributes is empty then
7:    single-node tree root, with label= most common value of Target_attribute
      in Examples
8: else
9:    A := the attribute from attributes that best classifies Examples
10:   The decision attribute for root:=A
11:   for each possible value, V_i, of A: do
12:      Add anew branch tree below root, corresponding to test A=V_i
13:      Let Examples_V_i be the subset of Examples that have value V_i for A,
14:      if Examples_V_i is empty: then
15:         Add a leaf node below the new branch with label= most common
            value of Target_Attribute in Examples
16:      else
17:         add the subtree below this new branch
18:      end if
19:      call ID3(Examples_V_i, Target_attribute, Attributes-A)
20:   end for
21: end if
22: return Root
```

Chapter 4

Hyperball Algorithm (HBL)

4.1 Introduction to HBL Algorithm

Hyperball learning algorithms (HBL) [7] are a set of classification and clustering algorithms designed to work for both supervised learning and unsupervised learning. HBL divides supervised learning into learning from a fallible and from an infallible expert. It also provides methodologies for self supervised learning (autonomous learning). HBL algorithms are designed to work in a highly parallel manner, but they can still be utilized in sequential processing as well. HBL algorithms can also be implemented online so they evolve with changes made in their environment, although it is not within the scope of this thesis.

4.1.1 Background

HBL algorithms belong to the group of Instance Based Learning algorithms (IBL) [24] . IBL algorithms learn by storing the presented training data in a particular way. In testing mode, when a new instance (pattern) is encountered (from testing set), a search is done to extract a set of

similar related instances from the storage (memory), which is then used to classify the new instance. One advantage of IBL algorithms is their ability to construct a different approximation to target function for each distinct testing instance (pattern) [24]. This feature is advantageous when target function is complex and should be described by a collection of local approximations which have lower complexity.

One of the main disadvantages of IBL algorithms can be the high cost of the classification of new instances. In IBL algorithms, nearly all computation takes place at testing time, which can introduce a time complexity factor, especially when the classifier is dealing with larger sample sizes. Another disadvantage which applies to classifiers like KNN is that they consider all of the attributes associated with data when they try to retrieve similar instances from their memory. This could lead to the selection of non-similar instance when the label of data (target) is just dependent on a few attributes [24].

Among all learning algorithms of IBL (radial basis functions, case based reasoning, etc), KNN is most similar to HBL algorithm. Indeed, they both store training data and calculate a distance from a new instance (pattern) x_q to all saved individual train data to identify the class of the query pattern. However, HBL algorithm centers a secured-margin (ball) around each pattern before storing the pattern. The existence of a ball centered at pattern is helpful when testing instance x_q is located inside another previously stored patterns ball. It makes the classification faster and more efficient in a way since no distance calculation is required any more after the identification of a ball containing the pattern in question. This can happen by simply returning the category to which the ball belongs to as the classification decision instead of calculating all distances as in regular KNN algorithm. We need some mathematical definitions prior to describing HBL algorithm. These definitions are explained in the

next section.

4.1.2 Basic Definitions

The **Metric**: A set of points such that for every pair of points there is a nonnegative real number that is symmetric and satisfies the triangle inequality.

Def.1 Measure of distance: A real-valued function $d : SxS \rightarrow \mathbb{R}$ and $S \subset \mathbb{R}$ can be used as a measure of distance, provided that for all $x, y, z \in S$ it has the following properties:

$$\begin{aligned} d(x,x) &= 0 \\ d(x,y) &= d(y,x) \\ d(x,z) &\leq d(x,y) + d(y,z) \\ x \neq y \quad then \quad & d(x,y) > 0 \end{aligned} \tag{4.1}$$

Examples of such metrics can be the Manhattan distance and Euclidean distance.

Def.2 Metric space: The configuration $< S, d >$ where S is a set of points, and d is some measure of distance between them [25].

Measuring distances between subsets of S will suit HBL better. One of the simplest subsets of S is a *ball*.

Def.3 Ball: A *ball* $B(c, r)$ of radius $r \geq 0$ around the point $c \in S$ is the set

$$\{x \in S \quad | \quad d(c,x) \leq r\} \tag{4.2}$$

The point c is called the center of the ball B.

Computing Distances between sets

Considering any two non-empty sets $A, B \subset S$, a function $D(A, B)$ needs to be defined to measure distance between A and B while satisfying conditions specified by 4.1. Let d be a chosen function for measuring distance between points of space S.

Def.4 Distance between a point and a set:
Let $< S, d >$ be a metric space and let $A \subset S$ be a non empty set. A distance between a point $x \in S$ and A, is defined as

$$\delta(x, A) = inf\{d(x, a) \quad | \quad a \in A\} \tag{4.3}$$

where $\delta(x, A)$ is the distance between x and a point $a \in A$ closest to x and inf stands for infimum value. By using the definition of infimum above, a hyperball is defined as follows:
Def.5 A **hyperball** of radius r>0 around a set $A \subseteq S$ is the set

$$\{x \in S \quad | \quad \delta(x, A) \leq r\} \tag{4.4}$$

Such a hyperball has the following properties:

- If the set $A \subseteq S$ consists of a single point, the hyperball reduces to a ball around a point.
- If the radius reduces to zero, the hyperball will be equivalent to the center set.

Def.6 Pseudo distance between two sets: Let $< S, d >$ be a metric space and let $A, B \subset S$ be two non empty sets. A pseudo-distance from A to B, denoted $\Delta(A, B)$ is given by:

$$\Delta(A, B) = sup\{\delta(a, B) \quad | \quad a \in A\} \tag{4.5}$$

It means that the pseudo-distance between A and B is the distance from the most distant point of A (from B) to B (*sup* stands for supremum value).

Def.7 Distance between two sets: Having calculated pseudo distance from A to B and from B to A, the larger of these two values is the distance between two sets or $D(A,B)$.

$$D(A,B) = max\{\Delta(A,B), \quad \Delta(B,A)\} \tag{4.6}$$

We will use these definitions to describe the HBL algorithm. This definition is used in HBL algorithm to calculate distance between two sets. HBL benefits from the former last two definitions to calculate the distances between two sets.

4.1.3 Pattern Recognition and Object Classification

Learning can be categorized into two main groups: supervised learning and unsupervised learning. In supervised learning, training data consists of vector data input and desired output. In case of unsupervised learning algorithms, only a vector of data inputs is available. HBL algorithms can be applied to both supervised and unsupervised learning [7]. As this thesis uses HBL algorithms for supervised learning, main focus of this part will be on HBL's supervised learning algorithms.

HBL uses its knowledge bank, K being a set of different categories $K = \{C_1, C_2, ...C_k\}$. Each category is associated with a different target class. As HBL encounters a pattern which is to be classified, it searches its knowledge bank to see if it can classify the pattern (Algorithm 4). If pattern can be classified, HBL returns the category index of the classified pattern or zero if the pattern cannot be classified. In Algorithm 4, HBL checks to find out whether the query pattern is inside any balls from train-

ing set. If so then HBL returns the category of that ball as the classification decision (Lines 3-4 in Algorithm 4).

Algorithm 4 Hyperball algorithm for pattern classification.

```
1: function Classifier(P: Pattern, K:Knowledge Bank) returns k
2: for k in 1...card(K) do
3:    for i in 1...card(K.C(K)) do
4:       if D(P, K.C(k).B(i).P) ≤ K.C(k).B(i).r then
5:          return k
6:       end if
7:    end for
8: end for
9: return zero
```

It is important to note that balls belonging to different categories are altered by HBL algorithm in a way that they are mutually exclusive, so this algorithm will return the result only once.

4.2 Supervised Learning Algorithm

In supervised learning, whenever HBL encounters a pattern, it first centers a ball around the pattern, then it checks to see whether it can classify it based on patterns learned before. This process is done through HBL's Classifier. In case the pattern has not been learned previously, HBL inserts the ball containing the pattern into the category matching the target class of the pattern (Algorithm 5 lines 1 – 9).

As demonstrated in algorithm 5 , balls are then sorted in each updated category (in descending order of their radii) and then the algorithm tries to remove redundant balls. A redundant ball is a ball which is completely inside another larger ball when both balls belong to the same category. In this case, the inner ball is redundant and can be removed, ensuring that

Algorithm 5 HBL Learning Process: Supervised Learning

```
1: Procedure Learn(P:Pattern; r:Radius; idx:Category Index;) returns
   CatIndx
2: if k(Number of Categories)<idx then
3:    k=k+1;
4:    Create new empty category C(k);
5:    Insert C(k) into Knowledge bank K;
6: else
7:    k=idx;
8: end if
9: Insert Ball B which contains P into K.C(k);
10: Sort balls in K.C(k) in descending order of radius;
11: Remove redundant balls from K.C(k) if any;
12: for n in 1..card(K) and n <> k do
13:    for i in 1.. card(K.C(n)) do
14:       if D(P,K.C(n).B(i).P) <= r then
15:          Remove ball K.C(n).B(i) from K.C(n);
16:       else if D(P,K.C(n).B(i).P)<= r+K.C(n).B(i).r then
17:          K.C(n).B(i).r=D(P,K.C(n).B(i).P)-r-ϵ;
18:       end if
19:    end for
20:    Sort balls in K.C(n) in descending order of radius
21:    Remove redundant balls from K.C(n) if any
22: end for
23: Remove empty categories from K if any
```

the balls B_i, $i = 1, 2, 3, ...n$ characteristic of a given category *CatIndx* either stand apart or only partially overlap (lines 10 and 11).

In the same way, a check is to be made whether balls in other categories conflict (are not mutually exclusive) with the ball just inserted into C_{idx}. If so the radii of these balls are reduced in order to eliminate the conflict. This procedure is done as follows: In line 14 it is checked to find out whether another pattern of a different class will be inside ball of new pattern P after insetting the new pattern P into its category. In case this condition happens then the inner pattern is removed from knowledge bank (i.e its category). If two patterns with different categories are having an intersection then according to line 15 radius of the pre-existing ball

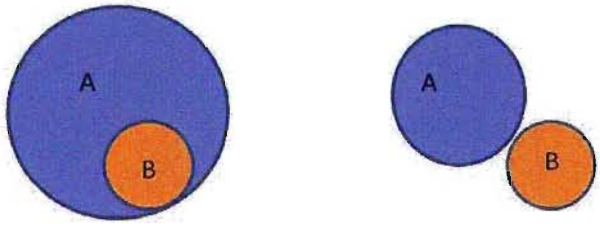

Figure 4.1: Ball Shrinking Procedure

will be reduced accordingly to make the balls mutually exclusive. This procedure is shown in Figure 4.1. If their radii shrink to zero, such balls are removed from the knowledge bank [7].

In case two balls which belong to two different categories are not mutually exclusive, then their corresponding radii should be altered in a way that they are mutually exclusive. This procedure will have a time complexity of $O(N^2d)$ for HBL according to Algorithm 5.

4.2.1 HBL Variants

In this part, we graphically demonstrate the possible issues with the original algorithm and how these issues are resolved. These graphics use Euclidean metric, while HBL works with any metric [7].

In Figure 4.2, HBL's classifier has already learned patterns P1 to P4, out of which $P2$ and $P4$ belong to the same Category (B), while P1 belongs

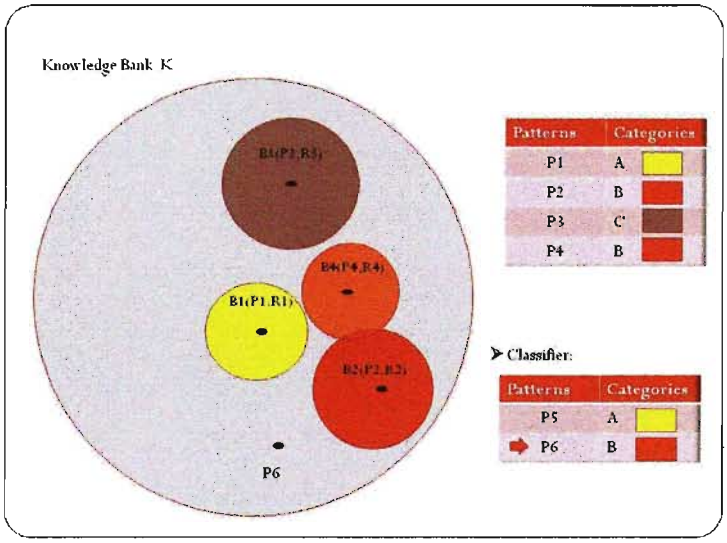

Figure 4.2: HBL Classifier

to category *A* and P3 belongs to category *C*. As it is also apparent in Figure 4.2, HBL's classifier is about to classify pattern *P6* which also belongs to category *B*. The classifier realizes that *P6* will not be inside any learned balls so according to the pseudo-code presented in algorithm 4 classifier will return zero which indicates that no matching category has been found. This can lead to poor generalization as there can be many patterns which cannot be located inside a learned ball. However, this problem can be resolved by calculating the distance of each pattern which is not inside any learned balls to all existing balls of all categories and choosing the category of closest ball as the classified category (Figure 4.3). The idea is similar to how KNN classifiers work with the difference being that in KNN, the distance between two patterns are considered whereas in HBL, the closest distance between a pattern and a ball is taken.

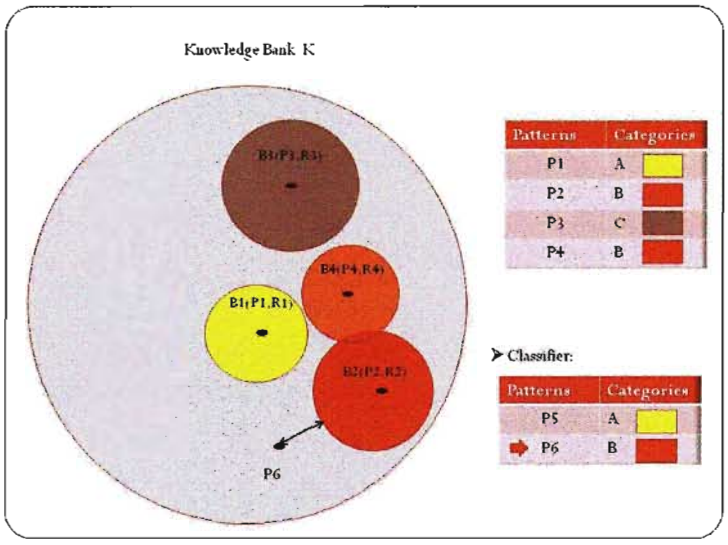

Figure 4.3: HBL Classifier: Second Iteration HBL_1

HBL_1

Now let's imagine the situation shown in Figure 4.4 where $P4$ is to be learned. The classifier realizes that $P4$ is inside $B2(P2, R2)$. As both $P4$ and $P2$ belong to the same category, the HBL classifier will not center a ball around $P4$. Now consider $P5$ as the next pattern which belongs to a class different to P2 and P4's class. As we can see in Figure 4.5, after the classifier centers a ball around the pattern and inserts it into its category, it needs to shrink $B2(P2, R2)$ to avoid balls containing patterns P2 and P5 overlap. This can lead to a problem in a way that after B2 shrinks it does not hold P4 any more, meaning classifier can forget pattern p4 by mistake.

In order to overcome the above problem, HBL's learning algorithm in algorithm 4 is modified in such a way that whenever a pattern is inserted into a category and is found inside another pattern's ball of the same

category, a ball is inserted around the pattern. Now, if after learning proceeds other balls shrink, then, this pattern is assured to remain inside a ball and will not be forgotten. This process has been depicted in Figures 4.6 and 4.7. Another modification which is done to algorithm 4 pertains to the situation in which a ball of a category is inside another ball a of different category. In this case according to algorithm 5, the inner ball is removed from its corresponding category, which can cause the classifier to forget the pattern it has learned previously. The solution which is provided in this thesis is to shrink both balls accordingly to avoid losing information about the inner ball (Lines 14-16 in algorithm 6). After applying these modifications to algorithm 4. HBL learning algorithm is presented in algorithm 5.

Algorithm 6 HBL: Modified Learning Process

Procedure Learn(P:Pattern; r: Radius; idx: Category Index)returns CatIndex

1: **if** k(Number of Categories)< idx **then**
2: k=k+1;
3: Create new empty category C(k);
4: Insert C(k) into Knowledge bank K;
5: **else**
6: k=idx;
7: **end if**
8: Insert Ball B which contains P into K.C(k);
9: Sort balls in K.C(k) in descending order of radius;
10: Remove redundant balls from K.C(k) if any;
11: **for** n in 1..card(K) and $n \neq k$ **do**
12: **for** i in 1.. card(K.C(n)) **do**
13: **if** D(P,K.C(n).B(i).P) <= r **then**
14: shrink both B and K.C(n).B(i) according to following:;
15: $K.C(n).B(i).r = \frac{D(P,K.C(n).B(i).P)}{2} - \epsilon$;
16: r= $\frac{D(P,K.C(n).B(i).P)}{2} - \epsilon$;
17: **else if** $D(P,K.C(n).B(i).P) <= r + K.C(n).B(i).r$ **then**
18: K.C(n).B(i).r=$D(P,K.C(n).B(i).P) - r - \epsilon$;
19: **end if**
20: **end for**
21: Sort balls in $K.C(n)$ in descending order of radius
22: **end for**
23: **for** n in 1..card(K) **do**
24: Remove redundant balls from K.C(n) if any
25: Remove empty categories from K if any
26: **end for**

Majority Voting (HBL_2 And HBL_3)

Here we present another improvement to HBL algorithm: Instead of making decisions based on the closest ball to the pattern, we can take the majority voting using k nearest balls to the pattern. If there is a tie, then balls which have a greater radius will be superior. This is because the pattern which has a greater radius is less likely to be closer to patterns which belong to other categories. The choice of k, of course will be problem de-

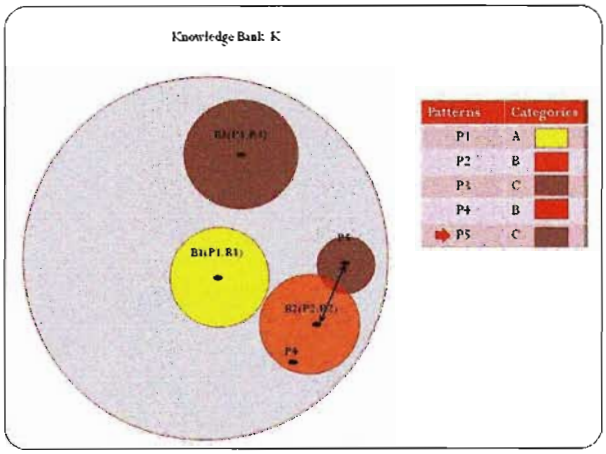

Figure 4.4: HBL Classifier

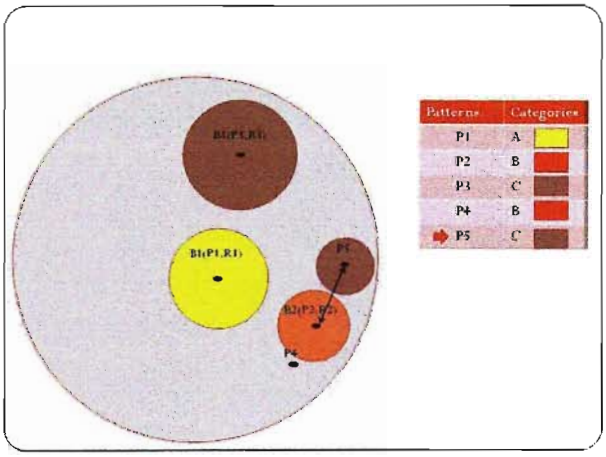

Figure 4.5: HBL Classifier

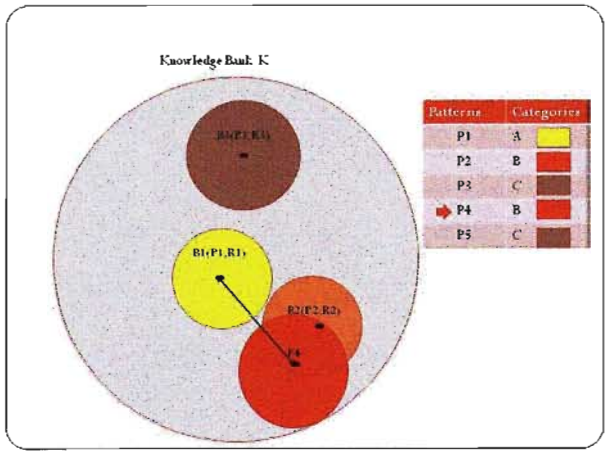

Figure 4.6: HBL Classifier

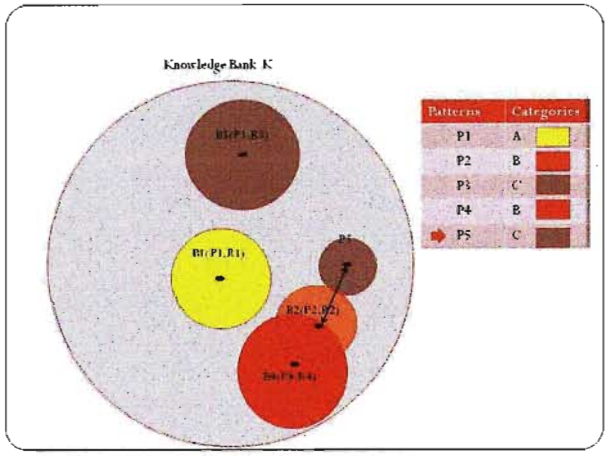

Figure 4.7: HBL Classifier

pendent. This methodology will resemble our HBL_2 algorithm. HBL_3 is a slightly modified version of HBL_2 in the sense that a weighted distance calculation is replaced by the normal Euclidean distance formula. This can be accomplished by the following rule :

$$|S| \equiv \{x \quad | \quad \Sigma_{i=1}^{k} w_i \delta(v, f(x_i)) \quad is \quad maximized\} \tag{4.7}$$

Where,

$$w_i = \frac{1}{d(x_q, x_i)^2} \tag{4.8}$$

Chapter 5

A Note on KNN and HBL

5.1 Introduction and Literature Review

Out of all of the classifiers we have discussed in this thesis, KNN algorithm is the most similar to HBL algorithm. Both of the algorithms store training set patterns to classify patterns from testing set. Moreover, both algorithms make classification decisions based on distance calculation as a similarity measure and by looking for the closest neighbour (most similar pattern) in their training sets. HBL algorithm is, however, different to basic KNN algorithm in that it centers a ball around each pattern. During the training phase, HBL tries to adjust and tune the radii of the balls around each pattern to make sure that the balls belonging to different categories are mutually exclusive. HBL also calculates the distances to balls centered around patterns instead of calculating the distances to exact patterns. This makes HBL similar to Radial-basis-based methods [26] and Gaussian Mixture models [1] which are out of this works focus.
The above explanation highlight the importance of completing a literature review on variants of KNN algorithm to see whether or not HBL is similar to any variants of KNN. As both of these algorithms make decisions based on distance measure calculations, we have included a literature review of

available distance measures in the next part.

5.1.1 A Note On Distance Measures

Recall from the definition of a metric space given in HBL chapter (Chapter 4) in equation 4.1 any distance metric chosen for KNN should conform to all four criterion defined there.

One of the well known distance metrics is Minkowski [27] distance metric:

$$MD_p(q, x_i) = (\Sigma_{f \in F} |q_f - x_{if}|^p)^{\frac{1}{p}} \tag{5.1}$$

where F represents total features, q_f a feature of query example from testing set and x_{if} is *ith* pattern from training set. L_1 Minkowski distance (*i.e* $p = 1$) will be equivalent to Manhattan distance [28] and L_2 distance will be equal to Euclidean distance measure [28]. Choosing larger values for p will ensure greater weights for features regarding which the patterns differ the most.

There are other distance metrics which are more suitable for multimedia data (like image). Many of these metrics like Kullback-Leibler Diverage [29] and χ^2 statistic [30] rely on comparing color histograms of data. In such metrics, an image is considered as a gray scale histogram which has N levels. However Padraig Cunningham et.al in [31] argue that such metrics do not satisfy the symmetry requirement in equation 4.1. He also mentions that such measures are prone to errors because of existence of histogram level boundaries which can lead to showing a great difference between an image and a slightly darker copy of it [31].

Another metric measure which overcomes the above drawback is called Earth Mover Distance (EMD) [32]. This measure is based on the amount of work which is required to convert one image to another. Images in this measure are considered to be distributions in the sense that

one image is considered to be earth in the space and another image, as another distribution, which is considered to be a hole. Then EMD will be the minimum amount of work required, so the earth can fill the holes.

The last distance measure which we will discuss in this section is called compression based (dis)similarity [33]. The idea is that if two documents are very similar, then if we concatenate them and compress them again, the size we will get is not much greater than any of the compressed original single documents. This statement will not hold true when two documents are not similar. For this work Euclidean distance is used for HBL and KNN distance calculations, however many of these distance measures can be applied to HBL algorithm [7].

5.1.2 Basic KNN Overview And Its Drawbacks

Basically, KNN classification has two stages. In the first stage nearest neighbors are determined, and in the second stage the classification class is determined using these neighbours. The K nearest neighbors are selected based on a distance metric. There are many ways in which K nearest neighbors can be used to decide the class to which a query pattern q belongs. One of the most straightforward approaches is to use majority voting among the selected neighbors to find the class to which pattern belongs and assign it to the query pattern q (from testing set).

The basic KNN algorithm is very straightforward to implement, however it suffers from a few drawbacks [31]:

1. When datasets become very large, training set also becomes large so searching for the nearest neighbour pattern in training set can be computationally very expensive.

2. When the number of features for the data (data dimensionality) is high, then distance calculations can become very costly.

3. If data is associated with noise and erroneous labels then there is a possibility for deviation from optimum outcome and results.

The basic KNN algorithm with Minkowski distance measure has a complexity of $O(|D||F|)$ where D is the training set and F is set of features associated with data [31]. However, the computational complexity of EMD is $O(|D|n^3log(n))$ where n is the number of clusters [32].

As we can see above, both training and feature sets have a great effect on time and computational complexity of a distance measure. Research has mostly focused on improvements to overcome the aforementioned disadvantages of KNN by proposing methods and strategies to reduce the dimensionality of data and edit down the training set in the data [31]. Some of this research has led to the introduction of faster and more efficient KNN variants [34, 35, 36, 37]. We have tried to summarize some of these efforts in the next section.

5.1.3 Categories of KNN Algorithm Variants

There have been many studies regarding KNN algorithm in literature. To the best of our knowledge, most of the research done has been towards the goal of editing down the data in training set, reducing the number of features associated with data and alternative propositions to exhaustive search in basic KNN algorithm. Cunningham et.al [31] have highlighted four different strategies for speeding up nearest neighbor retrieval:

- Case-Retrieval Nets (CRNs) [38, 39]: One of the most widely used techniques for retrieval process. A network structure is formed by preprocessing the cases and it is used in the retrieval time. CRNs can be configured to return exact cases as KNN.

- Footprint Based Retrieval: A strategy used to speed up KNN. It is

made up of a two stage retrieval process which operates on a two level hierarchy [31]. First level corresponds to conducting a search in the case-based local regions which contain the target problem. Second level finds the closet case to this target problem in the same region [40].

- Fish and Shrink Method: This strategy exploits the triangular inequality property to make a scheme for case base to candidate neighbors formation. So this technique therefore requires the distance to be a true metric because of above reason. Another feature of this strategy is that it will ignore the cases which are very far from the query pattern [41].

- Fast KNN variants: According to [42], fast KNN algorithms try to reduce the required number of comparison while they also try to maintain original classification accuracy which could be reached when all number of comparisons were taken into account. "Comparison reduction" can be achieved by using different available methodologies:

1. **Prototype selection methods and transformations to other metrics:**

   These types try to apply a preprocessing mechanism to the training set to select a subset instead of applying the algorithm to the entire training set. This preprocessing mechanism can be based on the construction of a dissimilarity matrix which stores a range of similarities between prototypes [42]. Some researchers in [43, 44] have tried to transform the dissimilarity matrix into an Euclidean space and some others [45] have tried to use dissimilarities to represent each prototype by creating and assigning a feature vector

to each prototype. There are also other approaches used in prototype selection, Bandyopadhyay et.al. in [46] proposed an algorithm for reordering and sorting the patterns in training set which could reduce the required number of distance calculations.

2. **Working with original prototype space:**
   These types of algorithms work with original data (no transformation) and can be categorized as follows:

   2.1 **Tree based Algorithms:**
   Most of these algorithms use a branch and bound technique to partition the training set into regions, then a tree structure is used to speed up the search. The examples of these tree based structures are Kd-tree [47, 48], R tree [34], SS-tree [35], SR tree [36], FNA [37] tree.

   FNA tree is one of fast KNN variants which has been studied more according to [42]. In this algorithm, K-means algorithm is used to divide the training set into C subsets where each subset represents a node of the tree and is further decomposed to construct the whole tree.

   2.2 **Elimination Based Approaches:**
   These types of algorithms use a pruning rule derived from triangular inequality to avoid some comparisons to prototypes. In order to achieve this goal both tree structure and projection based approaches are employed [37, 49, 50, 51, 52].

2.3 **Approximate NN Search:**
These algorithms try to find an approximation to NN (nearest neighbour) instead of finding an exact NN (nearest neighbour) [53, 54].

HBL algorithm does use the techniques in tree based algorithms, neither uses any elimination based approaches nor tries to approximate the nearest neighbor so does not have any similarities to above methods.

5.1.4 Dimension and Noise Reduction

There are different variants of KNN which benefit from methodologies to reduce the dimensionality and noise associated with data in order to improve KNN performance. In the next two sections we have categorized these approaches.

Dimension Reduction

According to [31], dimension reduction techniques are mostly suitable to be applied to multi media data. Cunningham et. al also argues that dimension reduction can be achieved either by selecting a subset of data or by selecting a subset of features which describe the data. Feature reduction can be achieved by using methods like PCA (principal component analysis) [8] to transform data into a lower dimensional space or it can be used to discard some of the features. There are many techniques for feature selection. We have listed two of them as follows:
Filter methods in which before the actual execution of learning algorithm, strategies will be employed to remove irrelevant features from data. Furthermore, the selected subset of features will be used to train the algorithm. Example of such strategies are information gain (IG) which has been explained in the decision tree chapter and Odds Ratio (OR) [55]. OR calculates the ratio of odds of a feature from one class to another. This

method can be used to rank the features of data according to their odds value.

Wrapper techniques unlike filter methods will benefit from main learning algorithm to estimate the importance of features. This means that the wrapper searches the feature space to select a subset of features that maximizes the predictive accuracy of a classifier [56]. This can be achieved using two approaches: In forward selection, wrapper starts with an empty set of features based on the evaluation made by the classifier. In backward selection, all features are initially considered and wrapper tries to eliminate some features [31].

Noise Reduction

One of the disadvantages of KNN algorithm described in this chapter is that the algorithm may perform poorly if the data is associated with noise or erroneous labels. Early techniques for noise reduction includes case-based editing techniques. These techniques are known as methods which can be employed to achieve redundancy reduction and to remove noisy or corrupt cases from training set [57]. According to [31] editing methods can be divided into two main categories:

Competence Preservation Techniques [58] which correspond to redundancy reduction, and removing cases which do not contribute to classification competence. The next set of techniques are called competence enhancement techniques which correspond to noise reduction and removing corrupt patterns from dataset.

Examples of competence preservation techniques can be Condensed Nearest Neighbor (CNN) [59] in which any pattern which cannot be classified correctly is added to an initially empty set. Another example of this method is Selective Nearest Neighbors (SNN) which states that every

example in the training set should be closer to another example of the same class rather than to an example of a different class [60].

Edited Nearest Neighbors (ENN)[61] represents a strategy for competence enhancement in which it removes examples from the training set which do not agree to their k nearest neighbors. If multiple passes are done on training set, the algorithm is called Repeated ENN [62].

HBL algorithm has similarities to basic KNN algorithm but not to any of these variants which were tried to be categorized in this chapter. All these variants have tried to improve on of the issues and disadvantages associated with KNN (dimensionality, noisy data, sample size) HBL algorithm does not apply any of such preprocessing methods or algorithms so it has no similarities to these categories. For this reason, HBL algorithm is compared to basic KNN algorithm. In the next section we have compared the way each of these algorithms deal with Euclidean distance calculation as a metric measure.

5.2 HBL vs. Basic KNN algorithm

Given all different categories and variants of KNN algorithm, we will try to compare HBL algorithm to basic KNN algorithm. Then we will try to see if HBL has similarities to any KNN variants discussed in the previous section.

Given two single featured patterns $P1$ and $P2$ and the Euclidean distance measure KNN calculates the distance as follows:

$$D_k = \sqrt{P_1^2 - P_2^2} \tag{5.2}$$

Now if we consider HBL algorithm, it centers two balls $B1$ and $B2$ around $P1$ and $P2$ and then tries to tune radii of these balls i.e $r1$ and $r2$ while in training mode. Now HBL calculates the distance as follows:

$$D_h = max\{\sqrt{P_1^2 - P_2^2} + (r1 - r2), \sqrt{P_1^2 - P_2^2} + (r2 - r1)\} \tag{5.3}$$

By comparing equations 5.2 and 5.3 we can conclude the following:

$$D_h = max\{D_k + (r1 - r2), D_k + (r2 - r1)\} \tag{5.4}$$

The above equation will ensure that if the radii of all balls in HBL is set to zero, then HBL and basic KNN algorithm calculate distances in the same way. Another difference of HBL to KNN algorithm is in the way they try to classify a query pattern. HBL algorithm first checks to see if the pattern falls within the radius of any ball around patterns in training set. If so, it will return the class of that pattern in training set. If not, HBL uses the distance calculation to find nearest neighbors. Yet another difference of HBL algorithm to basic KNN algorithm is that HBL algorithm performs a partitioning of training set into the number of classes associated with data in a way that all patterns belonging to similar classes will be in one partition. This will not be as fast as some KNN variants (like FNA)

which use tree structures and partition data on each node of the tree. The inspiration behind this design in HBL is that if algorithm is implemented in a way to work in parallel, then all of the categories (partitions) can be searched in parallel to speed up the algorithm.

A Note on Efficiency of KNN and HBL Algorithms

According to KNN algorithm given in Algorithm 2 and considering N as the cardinality of a sample size and d as dimensionality of data and Euclidean distance calculations, KNN algorithm has a running time of $O(Nd)$. More over, the only space required is to store the training set data as no computation is taking place in training mode.

Now considering HBL, we can give an upper bound of $O(N^2d)$ for the training mode . Considering the complexity as a factor, in training mode KNN just stores the patterns in its memory whereas HBL performs partitioning of data. This partitioning how ever adds a complexity factor to the classifier but it is beneficial to HBL algorithm as it can improve its performance in testing mode.

When it comes to testing mode, KNN algorithm has a time complexity of $O(Nd)$ as explained earlier. HBL in worse case condition has the same time complexity as for KNN but that can happen when data in testing set has the most distance to data in testing set. This is true because when testing set data is very distant to data presented in training set, then testing set patterns are less likely to be included in any balls of training set (advantage factor of HBL). As normally data which belong to a class of data have similarities this is less unlikely to happen. Even if one pattern is located in a ball of training set patterns, then HBL has an advantage in classification speed while in testing mode.

Chapter 6

Experimental Setup and Data Sets

In order to compare the performance of HBL to other algorithms, we are using the accuracy of each of these algorithms as a comparison criterion. By accuracy, we mean the ratio or percentage of correct classification of unseen instances from testing set to all of the instances available in test set (classification accuracy).

In this work, three different experiments are conducted to compare the accuracy of HBL to other classifiers. For all experiments, a set of nine benchmark data sets is selected from the UCI machine learning repository website [63]. Data sets are chosen in a way that they can be categorized as small to medium in sample size and dimensionality. Large datasets were excluded from this work because of resource problems. Each data set can be considered to be a distinct problem in error space upon which classifier generalizations accuracy are tested.

The first experiment compares classifiers when sample size is selected to be variable in each data set. In the second experiment, a similar comparison is made when the dimensionality of data varies. The last experiment compares classifiers based on their agreement to target values. In this

experiment, Rprop algorithm, HBL_1, HBL_3 along with KNN and ID3 are applied to benchmark datasets while their corresponding confusion matrices are recorded. Confusion matrices then are then used in Cohen's Kappa test of statistics [9] to determine the level of agreements between classifiers decision and actual targets.

In all experiments, each classifier is trained and tested using k-fold cross validation technique (k=10) [64, 65] using 30 runs. In this method all patterns in a dataset are divided randomly into k equally sized folds. In each pass (the number of passes are equal to k), $k-1$ folds are used for training and one fold is used for testing. The generalization accuracy as the percentage of correct classification (classification accuracy) is calculated in each pass and then averaged over all k passes for the last two experiments.

A statistical significance test (t-test) [66] is also utilized for the last two experiments which tests whether differences of the means is not due to randomness. However, as explained in the last paragraph, Kappa test of statistics is used in the last experiment to measure the level of agreement between the classifiers output and target values in data.
The remaining parts of this chapter are organized as follows: The next section briefly describes the benchmark data sets which are used in these experiments, then each experiment is explained separately in a section along with the utilized experimental setup for each classifier.

6.1 Benchmark Data Sets

As we explained above, nine main data sets are selected for all three different experiments. These datasets are shown in the Table B.1.

Iris data set, is a data set which has been used by many researchers in the field of pattern recognition [67, 68, 69]. Iris attributes describe the

length and width of the sepal and petal in each observed iris plant. The data set contains three classes of fifty instances each, where each class belongs to a type of iris plant. One of these classes is linearly separable from the other two, while the other two are not linearly separable.

Table 6.1: Benchmark Data Sets

Data sets Features				
DB-NAME	Attributes	#Patterns	#Classes	Attr-types
Iris	4	150	3	Real
Pima Indian	8	768	2	Integer,Real
Connectionist benchmark	60	208	2	Real
Glass	10	214	6	Real
Wine	13	178	3	Real
Parkinson	23	197	2	Real
Zoo	17	101	7	Categ,Int
Breast Cancer	10	699	2	Integer
Musk V2	168	6598	2	Integer

The **PimaIndian** data set [70], gathered by the National Institute of Diabetes and Digestive and Kidney Diseases, includes data from 768 females all at least 21 years old. All different eight attributes correspond to a different medical measure for each person. Pima data set is known as a data set with highly nonlinearity behaviour [70].

In the **connectionist benchmark** data set, the purpose is to train a classifier to discriminate between sonar signals bounced off of a metal cylinder and those bounced off of a roughly cylindrical rock. There are 60 attributes and 60 instances associated with this data set.

Next, the **Glass** data set, with seven different classes of output data and a small sample size of 214 instances, is a database which was studied with the idea of acquiring capabilities to detect the type of glass at the scene of a crime. This is desirable so that the glass which had been left there could be used as an evidence. Each instance in this data set is obtained from 10 different measurements, hence, it has 10 dimensions.

The **Wine** data set, is the result of a chemical analysis of wines grown in Italy and derived from three different cultivars. This data set has 13 attributes associated with it [71].

The **Parkinson** data set, is composed of a range of biomedical voice measurements from 31 people, out of which 23 have Parkinson's disease [72].

Zoo data set, result of 17 different measurement for 7 different classes of animals. This data set which is simpler as attributes are often boolean values is created artificially.

The **Breast Cancer** [73, 74, 75] dataset is obtained from the University of Wisconsin Hospital. It has consisted of 699 instances with 10 attributes. It also has some data with missing values.

The **Musk Version 2** data set, describes a set of 102 molecules out of which 39 are judged by human experts to be musks and the remaining 63 molecules are judged to be non-musk. Because bonds can rotate, each molecule can have many conformations (shape). This many to one mapping problem is been solved by generating low energy level conformation and extracting a feature vector for each conformation. There are 166 features associated with each molecule, out of which about 161 features are called " Distance Features".

6.2 Experimental Setup

6.2.1 Data Preprocessing

Data for all classifiers are standardized using the functions and tools available in Matlab R2009. Inputs and targets of data are standardized in a way that they have zero mean and unity standard deviation. For data

sets which have a missing value in their data, Matlab preprocessing tools are employed to transform each row of data containing a missing value into two rows that encode that same information numerically. This has been done by simply copying the first row containing missing values and replacing those missing values by the mean of that row. The second row can be a "zeros" "ones" map indicating if the original data at that position was known (a one) or if it was a missing (Zero).

Implementation

Matlab 2009 tool boxes are employed for the implementation of all classifiers except for HBL algorithm which has been implemented in JAVA 1.6 language. For Neural Network variants toolboxes of Matlab were used to save implementation time, however ID3, and KNN were hard coded in Matlab. HBL being as a new language needed to be hard coded as well. The reason HBL was not implemented in Matlab was due to the use of data structures which HBL used. The parameter setting and approaches for each classifier is summarized in the next page.

Neural Networks

For neural networks classifiers (back propagation algorithm and its variants), a hill climbing approach is followed in order to decide the structure (hidden neurons) of each network. In this approach, we simply try different numbers of hidden nodes from one to one third of the number of inputs to that network [76].

In order to decide the best structure, data is passed through fifteen times. When the structure has changed, the weights are reinitialized. For each structure, classification accuracy has been saved. At the end, the average best network with the best classification accuracy and the

least number of hidden units is selected. If there is a tie between two or more networks, the network with minimum number of hidden nodes is selected. Weights in the neural network are initialized using Nguyen-Widrow technique [77].

After the structure has been selected, the network is trained independently using each different algorithm. Different epoch values, validation failures (for the last two experiments), momentum and learning rates were tried after the structure was chosen for a neural network. Table 6.2 summarizes the parameters used for ANN variants.

Table 6.2: ANN Parameter Values

Algorithm	Max Parameter Description	Parameter Value
Gradient Descend	Number of EPOCHs	5000
	Validation failures	200
	Momentum	0.60
	Learning Rate	0.20
	Minimum Gradient (GDM)	1e-8
	Error Goal	0.001
Levenberg-Marquardt	Memory and Speed Tradeoff Factor	1
	Initial MU	0.001
	MU Decrease Factor	0.1
	MU Increase Factor	10
	Maximum MU	1e10
Quasi-Newton	Search Line Method	1-D minimization
	Lower Limit on change in Step Size	0.1
	Upper Limit on change in Step Size	0.5
	Maximum Step Length	100
	Minimum Step Length	1.0e-6
Resilient BP	Increment to Weight Change	1.2
	Decrement to Weight Change	0.5
	Initial Weight change	0.05
	Maximum Weight Change	60.0

Table 6.3: Structure of ANN, KNN, and HBL Algorithms

DB-NAME	ANN Structure	#K (KNN)	#K (HBL)
Iris	4-4-3	3	3
Pima Indian	8-26-2	21	21
Connectionist benchmark	60-24-2	9	13
Glass	10-28-6	9	9
Wine	13-25-3	5	8
Parkinson	23-17-2	11	7
Zoo	17-16-7	8	14
Breast Cancer	10-9-2	7	5
Musk V2	168-23-2	19	16

10 fold cross validation (CV) technique has been used to train each network. CV is employed in such a way that data is randomly divided into 10 folds. Each time, 8 folds are used for training, one fold for validation and one fold for testing the data. The process will continue until all 10 folds have been used for testing data. Then, the accuracy is averaged after the training/validation/testing has been completed 10 times. It is important to note that in each step of CV, the network is reinitialized and the previous trained network is discarded. The whole process is repeated for 30 runs. Training is terminated if one of the following conditions occurs:

- If epochs of 5000 is reached or,
- If maximum failures (increase in performance) is reached in validation set or,
- If the minimum gradient is met (gradient descent based algorithms) or,
- If Mean square error goal is met.

Selected structure of ANN, KNN and HBL algorithms are given in Table 6.3. As problem sizes are small to medium ANNs are chosen with

only one hidden layer. For ANNs, the selected structure is given as #Input-nodes-#Hidden nodes-#Output-nodes.

Dtree algorithm constructs a full tree according to the input data set. The best structure is chosen according to the classification accuracy. Data is been divided using 10 fold cross validation. Each time, 9 folds are used for training data and one fold for testing the accuracy. Results are then averaged over 10 attempts. ID3 is used just as an accuracy measuring tool and no pruning technique is been utilized.

HBL, KNN

HBL and **KNN** have similarities in implementation. Both algorithms calculate distance in Euclidean space. HBL can be applied to other metric spaces like Manhattan, Minkowski, etc. but as Euclidean space is the most known metric space, we employed this metric in this work. For HBL algorithm, $\epsilon = 0.0001$ is decided as the smallest possible space between two balls. This value is decided by using trial and error. Values both greater and smaller than $\epsilon = 0.0001$ were tried while data was passed through for 30 runs. ϵ value is decided experimentally in a way that values of $1, 0.5, 0.25, 0.1, 0.01, 0.001$ are tried before choosing the above mentioned value. To select K (optimum number of neighbours) for HBL and KNN, data is passed through 15 times and then the best value of K is selected based on obtained classification accuracies on testing data. Each time data is passed through, different values for K are tried between 1 to 30.

Both KNN and HBL algorithms use majority voting technique over selected K nearest neighbours to classify the data. In case there is a tie in majority voting, then the output class is decided based on a random selection.

Chapter 7

Experiments and Results

This chapter provides both the experimental analysis and results of this thesis. A comparison of the various classifiers is presented for each of the three experiments carried out.

Table 7.1: Abbreviations used in this section

Abbreviation	Category	Description
ANN	–	Artificial Neural Networks
LM	Neural Networks	Levenberg-Marquardt
GDM	Neural Networks	Gradient Descent with Momentum
BFG	Neural Networks	Broyden, et. al. Quasi Newton
RP	Neural Networks	Resilient Propagation
ID3	Decision Tree	–
IBL	—	Instance Based Learning
KNN	Instance Based Learning	K Nearest Neighbour
HBL_1	Instance Based Learning	Hyperball (K=1)
HBL_2	Instance Based Learning	Hyperball (k greater than 1)
HBL_3	Instance Based Learning	Hyperball (Weighted Distance)
0.25	–	25% Sample Size
0.50	–	50% Sample Size
0.75	–	75% Sample Size
1.00	–	Full Sample Size

For simplicity, summary tables are provided while the detailed results are available in Appendix B. For clarity, Table 7.1 provides a summary of the various abbreviations used in this chapter.

7.1 Experiment 1: Sample Size Effect

In this experiment, HBL is compared to other algorithms when different proportions of sample size are chosen for each dataset. The purpose of this experiment is to evaluate and compare the performance of HBL to other classifiers in terms of classification accuracy as a function of sample size. First we review some works found in the literature that studies the effect of sample size on classification accuracy.

Leshno *et. al* [78] studied the effect of training data size and the complexity of the separation function on neural network classification. Perceptron and feedforward network with a single hidden layer were used in this work. They employed two class data sets with two input variables, without noise but with different learning technique. They concluded that neural networks are a better choice than other statistical models because of their flexibility and improving learning capability.

Y. S. Kim [79] compared neural networks and decision trees to linear Regression methods. This study varied the number of independent variables, the types of independent variables, the number of classes of the independent variables, and the sample size. RMSE (Root Mean Square Error) was used as the metric. This study showed that linear regression algorithms are superior to ANN and decision tree for continuous independent variables. Kim also showed that ANN is the best when the number of categorical variables is at least two, and when independent variables are continuous and categorical.

Margarita Sordio [80] studied the effect of sample size on classification accuracy using a dataset having approximately 8500 patterns on Support Vector Machine (SVM), Decision trees, and Naive Bayes techniques. Sordio showed that the size of training set and the classification accuracy are correlated. Sordio also showed that when datasets are small all algo-

rithms perform good but as the number of sample size increases, SVM shows a very good improvement in performance. Sordio's work inspired the first experiment in this thesis, and the pseudo code presented below in Algorithm 7 is a slight modification of Sordio's algorithm that is employed in the first algorithm. Sordio used one dataset and increased the sample size one by one. In this thesis, we start with a quarter of the sample size and each time add 25% to the previous sample size until we achieve the full sample size.

Algorithm 7 First Experiment: Varying sample size in the data set

1: **for** each classifier **do**
2: **repeat**
3: Define selected set as percentage of the total number of cases in dataset (ranges: 25%, 50%, 75%, and 100%).
4: Train classifier with current set using 10-fold cross validation.
5: Evaluate the performance of the classifier as average correct classification rate.
6: **until** all cases in a dataset are used
7: **end for**

Once the data partitioning is done (at random), each classifier is trained with each partition independently. A 10-fold cross validation technique is used to train each classifier. The data sets used in this experiment have been divided into two groups. The first group consists of data sets which have smaller sample sizes (fewer than 500) or a smaller number of attributes (fewer than 50). The data sets in this group are Iris, Glass, Zoo, Wine and Parkinson disease dataset. Data sets in the second group are data sets with a sample size greater than 500 or with number of attributes greater than 50. These data sets are Musk, Pima Indian Diabetes, Breast Cancer and Connectionist data sets. These categories are shown in Table 7.2.

Table 7.2: Data Set Categories

First Category	Second Category
Iris	Musk
Glass	Pima
Zoo	Breast Cancer
Wine	Connectionist
Parkinson	

Table 7.3: Effect size table guide

Effect Size Value	Strength Interpretation
0.0	
0.1	
0.2	Small
0.3	
0.4	
0.5	Medium
0.6	
0.7	
0.8	Large
0.9	
1.0	

A statistical paired t-test is applied to classifiers to measure if the obtained results are significant within a 95% confidence interval. Moreover, an effect size [81] value is calculated for each comparison to measure the strength of the relationship between classifiers. The interpretation of effect size value is presented in Table 7.3. Since our focus is to compare the performance of HBL with other classifiers, first an inter-HBL comparison is performed to select the best HBL variant. Once this is done the best HBL variant is individually compared to each of non-HBL classifiers. For clarity, a summary of the inter-HBL comparison and HBL versus other classifiers is given below. However, the detailed experimental results are presented in Appendix B.

In order to compare different classifiers, we have used a two-variant comparison. This comparison is made in the following manner:
Classifier X is considered to be better than *Classifier Y* if:

- Classifier X has a greater mean (considering classification percentage) when compared to classifier Y.
- Paired t-test results show that the obtained results are significant.
- This comparison has an effect size of at least medium (i.e effect size ≥ 0.3).

If all of the above conditions are satisfied, then classifier X is considered to be better than classifier Y and it can be said that algorithm classifier X wins over classifier Y.

7.1.1 Inter-HBL Comparison

In this part three HBL variants are compared to each other and the best variant is then selected for comparison to other classifiers. Furthermore, we have enumerated the number of times a variant of HBL algorithm wins over another variant for all data sets.

The results provided in Table 7.4 show that HBL_1 and HBL_2 are not significantly different in 5 cases ($\frac{5}{9}$ = 55% of cases) when 25% and 75% of the sample size is selected. However when the full dataset is considered, HBL_2 wins more times than HBL_1 ($5 - 1 = 4$ times). The results from Tables 7.4-7.6 also confirm that HBL_3 algorithm outperforms the other two variants in more cases when compared to HBL_1 and HBL_2. So according to these observations, HBL_3 algorithm is considered the best HBL variant and is selected for comparison with other classifiers in this thesis. Therefore, in our comparisons we will use the name HBL for HBL_3 from this point.

Table 7.4: HBL_1 vs. HBL_2 Comparison

HBL_2 has a higher number of wins when compared to HBL_1

Proportions Of Data	HBL_1 Wins	HBL_2 Wins	Not Significant
0.25 Size	1	3	5
0.50 Size	1	7	1
0.75 Size	0	4	5
Full Size	1	5	3

Table 7.5: HBL_1 vs. HBL_3 Comparison

HBL_3 wins over HBL_1 in all levels of sample size selection.

Proportions Of Data	HBL_1 Wins	HBL_3 Wins	Not Significant
0.25 Size	1	7	1
0.50 Size	0	9	0
0.75 Size	0	8	1
Full Size	1	9	0

Table 7.6: HBL_2 vs. HBL_3 Comparison

HBL_3 dominates HBL_2 in all levels of sample sizes

Proportions Of Data	HBL_3 Wins	HBL_2 Wins	Not Significant
0.25 Size	7	0	2
0.50 Size	6	0	3
0.75 Size	7	0	2
Full Size	9	0	0

7.1.2 HBL vs. KNN

The results summarizing this comparison are shown in Tables 7.7 and 7.8. Comparison results of HBL to first category of data sets is shown in Table 7.7. Recall from Table 7.2, this category comprises the data sets with smaller (< 500) sample size or with smaller (< 50) number of attributes. The obtained results show that at 0.25 size, both HBL and KNN algorithms perform equally. They are not significantly different for 3 data sets and each have only one win. As the sample size increases, HBL wins more number of cases (has higher classification accuracy in more data sets). The

reason for this behaviour can be in the number of samples: As data sets in this category are small, when 0.25 of sample size is considered, only a few samples are selected. This can make the classification task easier, so both algorithms perform equally. As more number of samples are introduced to the classifiers, complexity of problem also increases. With this increase to the problem complexity, HBL algorithm outperforms KNN algorithm. One potential reason for HBL wins can be in weighted distance metric which HBL employs to calculate distances. Weighted distance calculation is suggested as an improvement over Euclidean distance calculation for Instance Based Learning (IBL) algorithms [82]. Another reason for HBL to outperform KNN can be in HBL Algorithm's structure: Before HBL seeks for a close neighbour in training set, it looks to see whether if testing pattern is inside any training pattern ball. When sample sizes are small, aforementioned condition is less likely to happen so the algorithms behave equally.

HBL comparison to KNN in second category in Table 7.8 also confirms the above statements: In the second category as the number of samples (or attributes) are greater, HBL outperforms KNN even when 0.25 of size is selected. Figure 7.1 shows the overall comparison of HBL to KNN algorithm when both categories of data are selected. It can be concluded that for the data sets employed in this thesis, HBL algorithm's classification accuracy outperforms as of KNN algorithm's.

Table 7.7: HBL vs. KNN Group 1 Comparison:

At first, number of samples are very less so algorithms are equally performed. But as the sample size increases, HBL algorithm wins more than KNN algorithm.

HBL vs KNN	HBL Wins	KNN Wins	Not Significant
0.25 Size	1	1	3
0.50 Size	5	0	0
0.75 Size	3	0	2
Full Size	4	0	1

Table 7.8: HBL vs. KNN Group 2 Comparison:

In this category, HBL algorithm wins more than KNN algorithm

HBL vs KNN	HBL Wins	KNN Wins	Not Significant
0.25 Size	3	0	1
0.50 Size	2	1	1
0.75 Size	3	0	1
Full Size	3	0	1

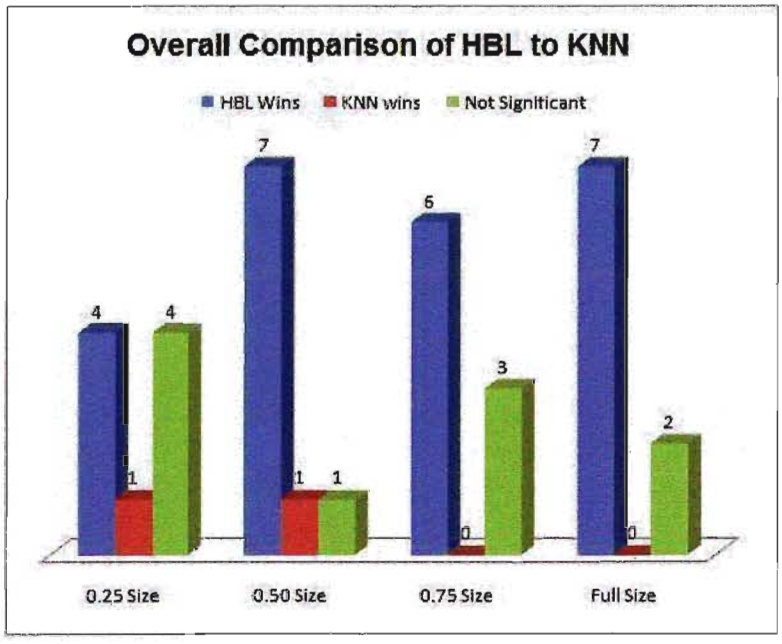

Figure 7.1: Overall Comparison of HBL to KNN: HBL outperforms KNN

7.1.3 HBL vs. Neural Network Variants

Tables 7.9 to 7.14 summarize the results of HBL versus ANN variants. Among the four different variants of BP neural networks which we have used in this work, LM and BFG are only applied to the first group of data sets as they can be computationally costly when the network size increases [1].

Table 7.9 summarizes results from HBL to LM comparison. The results provided in this Table show that when the sample size is small (0.25) HBL algorithm has more wins when compared to LM (4 wins for HBL and only one win for LM). But as the sample size changes and increases, LM algorithm shows equal results when it is compared to HBL algorithm. Although LM algorithm has more number of wins when it is compared to HBL at full sample size level, but it can not be concluded that either of these algorithms are performing better. This is also true in Table 7.10 where HBL algorithm is compared to HBL algorithm. From Table 7.10 we can conclude that both algorithms have performed almost equally. This result is repeated for Table 7.11 and Table 7.12.

Table 7.9: HBL vs. LM Comparison for First Category

HBL vs LM	HBL Wins	LM Wins	Not Significant
0.25 Size	4	1	0
0.50 Size	3	1	1
0.75 Size	2	2	1
1.00 Size	2	3	0

Table 7.10: HBL vs. BFG Comparison for First Category

HBL vs BFG	HBL Wins	BFG Wins	Not Significant
0.25 Size	2	2	1
0.50 Size	4	1	0
0.75 Size	3	2	0
1.00 Size	2	3	0

Table 7.11: HBL vs. RP Comparison for First Category

HBL vs RP	HBL Wins	RP Wins	Not Significant
0.25 Size	2	1	2
0.50 Size	3	2	0
0.75 Size	3	2	0
1.00 Size	2	2	1

Table 7.12: HBL vs. GDM Comparison for First Category

HBL vs GDM	HBL Wins	GDM Wins	Not Significant
0.25 Size	2	2	1
0.50 Size	4	0	1
0.75 Size	3	1	1
1.00 Size	2	2	1

In Table 7.13 HBL algorithm is compared to RP considering second category of data sets. As we have also shown in this table, this table shows that HBL and RP both have performed equally. However, when HBL is compared to GDM in Table 7.14, both algorithms start equally when only 0.25% of sample size is considered, but as fractions of sample size are added in next levels GDm algorithm outperforms HBL algorithm.

The overall results of comparison of RP and GDM classifiers to HBL algorithm are shown in Figures 7.2 and 7.3. In Figure 7.3, HBL starts better than RP but at full sample size level algorithms end up to be indifferent. The only variant of ANN algorithm that we can conclude that it has performed better than HBL algorithm is GDM. As its shown in Figure 7.2, GDM outperforms HBL algorithm as sample size increases. As three variants of ANN performed almost indifferent to HBL algorithm, we can conclude that in this work and for the datasets used in this work, HBL algorithm has performed equal to ANN back propagation variants.

Table 7.13: HBL vs. RP Comparison for Second Category

HBL vs RP	HBL Wins	RP Wins	Not Significant
0.25 Size	3	1	0
0.50 Size	1	3	0
0.75 Size	2	2	0
1.00 Size	1	1	2

Table 7.14: HBL vs. GDM Comparison for Second Category

HBL vs GDM	HBL Wins	GDM Wins	Not Significant
0.25 Size	1	1	2
0.50 Size	1	3	0
0.75 Size	1	2	1
1.00 Size	1	3	0

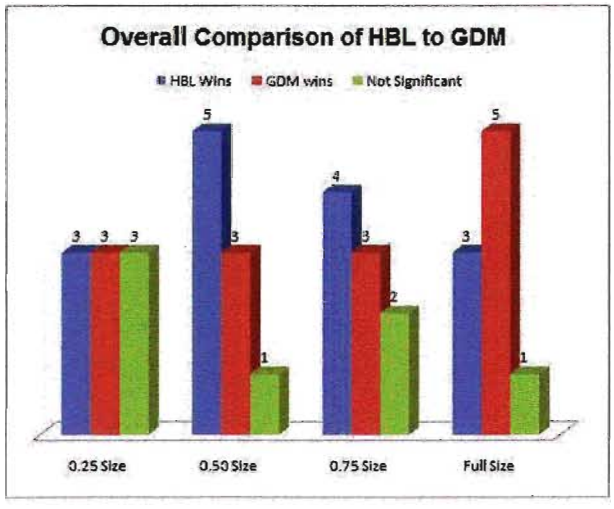

Figure 7.2: Overall Comparison of HBL to GDM

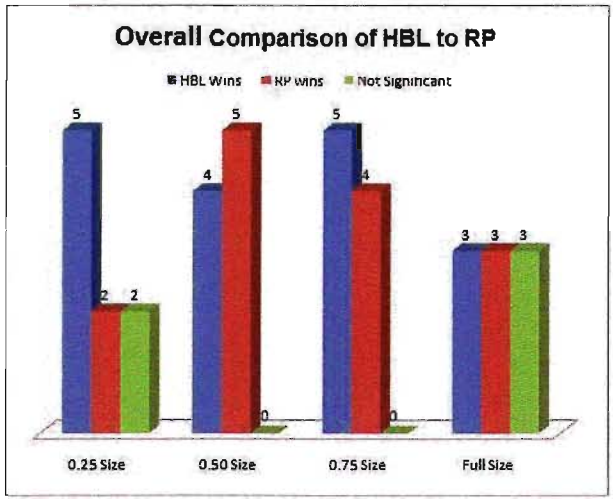

Figure 7.3: Overall Comparison of HBL to RP and GDM

7.1.4 HBL vs. ID3

The results from Table 7.15 show that HBL outperforms ID3 algorithm in all sample size levels (both categories of data sets). HBL has won in all 9 data sets for 0.25, 0.50, and 0.75 sample size values. When the full data set size is considered HBL has better results for 8 problem sets, whereas for in one data set the obtained results are not significantly different. The data set for which obtained results are not significantly different is Iris data set. In this data set two of three available classes are linearly separable which makes the classifying task much simpler. This is shown in Figure 7.4 using 1*st* and 3*rd* features of data set.

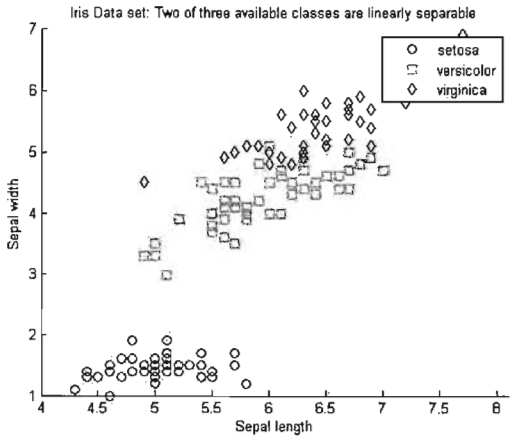

Figure 7.4: Iris data set:
Two of three available classes are linearly separable

Table 7.15: HBL vs. ID3 Comparison

HBL Algorithm outperforms ID3 algorithm. The only data set whose obtained result is not significant is Iris, in which two of three available class of outputs are linearly separable.

HBL vs ID3	HBL Wins	ID3 Wins	Not Significant
0.25 Size	9	0	0
0.50 Size	9	0	0
0.75 Size	9	0	0
1.00 Size	8	0	1

7.1.5 Experiment one's overall discussion

In this experiment HBL algorithm was compared to other algorithms when sample size in each data set was changing. The following conclusions can be extracted from the provided results:

1. In the first part of this experiment, an Inter-HBL comparison was done to select the best variant in HBL algorithm. HBL_3 outperformed other variants of HBL algorithm and so was employed

for later comparisons. The reason for HBL_3 to outperform other variants of HBL was that HBL_3 uses a weighted distance metric whereas remaining variants use Euclidean distance calculation.

2. Next when HBL was compared to Back Propagation variants the obtained results showed that HBL performs almost indifferent to all BP variants except for GDM which outperforms HBL algorithm.

3. HBL was then compared to back propagation neural network variants. ANN variants especially RP and GDM showed that they are growingly winning more times as the sample size is increasing in each level. The reason for that was sought in ability of back propagation variants to adjust and better adapt to different nonlinear environment [78]. HBL decreases in effectiveness when sample size increases, this could be because as the sample size increases chance of two balls of different categories to be inside each other increases. Recall from our discussions, in this case HBL algorithm shrinks the radii of balls in a way that they are mutually exclusive and this could lead to loss of information. .

4. Lastly HBL algorithm was compared to ID3 algorithm. In this comparison HBL algorithm outperformed ID3 algorithm in all levels of sample sizes. The only data set for which the comparing results was not significantly different at full size was Iris data set. For this data set it was shown in a Figure that two classes out of available three classes of data are linearly separable which makes classification task easier for ID3 algorithm.

7.2 Experiment 2: Data Dimensionality Effect

The main goal of this experiment is to study the impact of data dimensionality on classification accuracy. There has been much research performed about dimension reduction techniques and the effect of dimension reduction on classifier performance, we will first review some of these methods.

Plastria *et al.* in [83] studied the effects of dimensionality reduction on six two class data sets. They employed PCA, linear discriminant and introduced four new other techniques and showed that a good choice of technique and dimension can have a major impact on classification accuracy. Plastria deduces that there is no significantly superior dimensionality reduction technique which works the best for all problems in all error spaces. Others in [84] and [85] have also performed studies on dimensionality reduction techniques and its effects on classification error. In [84] a new algorithm called as classification constrained dimensionality reduction (CCDR) was compared to PCA algorithm for dimensionality reduction. K-nearest neighbour (KNN) algorithm's classification accuracy was used as a measure for this comparison. Raich *et. al* showed that when CCDR is used, KNN improves its performance by 10%. Mosci *et. al* in [85] studied the regularization property of Kernel Principal Component Analysis (KPCA). They showed that using KPCA and ordinal least squares method on projected data is equivalent to the use of spectral cut-off regularization where regularization parameter is equal to the number of principal components.

There are many dimensionality reduction methodologies available in literature. Independent Component Analysis (ICA) [86] is a method used in dimensionality reduction which attempts to find statistically independent components and transform the data onto those components. Random Projection method [87] is another method for reducing the di-

mensionality of data. The idea is that a projection matrix R of size $d * m$ can be carefully chosen in a way that $d \leq min(m,n)$ where m,n are the size of the original data matrix R. The matrix $D = RA$ resulted from the multiplication of these two matrices, and maintains approximately the same distances between data points. Another technique is called principal component analysis (PCA) [8, 88]. PCA consists of a procedure that is purposed to transform a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components. It is an orthogonal linear transformation which transforms the data into a new coordinate system in such a way that the greatest variance by any projection of the data comes to lay on the first coordinate which is called the first principal component. The second greatest variance upon the second coordinate and so on [8]. In this work PCA is used for dimensionality reduction because it is a well-known method, and a more straight forward technique for dimensionality reduction.

7.2.1 Algorithm And Experimental Procedure

As it has been explained earlier, the influence of an increase in data dimensionality on the classification accuracy of HBL algorithm is compared to other algorithms. Principal components will be computed for data using (PCA) and top (0.25, 0.50, 0.75, and 1.00) will be used to train each classifier:

7.2.2 HBL vs. KNN

Table 7.19 summarizes the obtained results for the HBL to KNN comparison. It is shown that HBL has 7, 6, 5 and 7 wins for 4 levels of dimensionality. These observations show that HBL outperforms KNN algorithm

Algorithm 8 Algorithm For Dimensionality Effect Experiment

1: **for** each classifier **do**
2: **repeat**
3: Use PCA to select top (ranges: 25%, 50%, 75%, and 100%) principal components of data.
4: Train classifier with current set using 10-fold cross validation.
5: Evaluate the performance of the classifier as average correct classification rate.
6: **until** all above percentages are used
7: **end for**

(the least value 5 is $\frac{5}{9}$ = 55.5% of cases). These obtained results ensure that HBL performs better than KNN in most cases.The reason for that can be different metric which HBL is benefiting from.

Table 7.16: HBL vs. KNN Comparison

KNN vs HBL	HBL Wins	KNN Wins	Not Significant
0.25 Dim	7	1	1
0.50 Dim	6	0	3
0.75 Dim	5	1	3
1.00 Dim	7	0	2

7.2.3 HBL vs. BP Variants

HBL algorithm is compared to RP in Table 7.17. The obtained results can not determine any of these two algorithms as winner except for the case in which 50% of top principal components are considered. In this level the obtained results from Table 7.17 shows that RP wins over HBL algorithm.

Table 7.17: HBL vs. RP Comparison

HBL vs RP	HBL Wins	RP Wins	Not Significant
0.25 Dim	3	3	3
0.50 Dim	3	5	1
0.75 Dim	2	4	3
1.00 Dim	3	3	3

7.2.4 HBL vs. BFG

HBL is compared to BFG variant of back propagation in Table 7.18. When 25% and 50% of dimensionality is considered, HBL algorithm has performed indifferent to BFG algorithm. However, when dimensionality of data is beyond 50%, BFG algorithm outperforms HBL algorithm.

Table 7.18: HBL vs. BFG Comparison

HBL vs BFG	HBL Wins	BFG Wins	Not Significant
0.25 Dim	2	4	3
0.50 Dim	2	4	3
0.75 Dim	2	5	2
1.00 Dim	3	6	0

7.2.5 HBL vs. ID3

HBL is compared to ID3 algorithm in Table 7.19. As it is shown in this Table, HBL outperforms ID3 algorithm in all dimensionality levels:
At 0.25 HBL wins 8 times, whereas ID3 wins over HBL only once. 7 wins for HBL at 0.50 and 0.75 levels will guarantee its dominance at these two levels. Finally when all the dimensions are used HBL still out performs ID3 by having 8 wins from a total of nine comparisons.

Table 7.19: HBL vs. ID3 Comparison

HBL vs ID3	HBL Wins	ID3 Wins	Not Significant
0.25 Dim	8	1	0
0.50 Dim	7	2	0
0.75 Dim	7	1	1
1.00 Dim	8	0	1

7.2.6 Overall Discussions

We can summarize the overall outcome of this experiment as follows:

1. In this experiment HBL algorithm was compared to two variants of ANN back propagation algorithm: RP and BFG variants. The results obtained from this experiment show that both of this variants performed indifferent to HBL algorithm. Moreover when the full dimensionality of the problem set is considered, BFG algorithm outperforms HBl algorithm.

2. HBL algorithm outperformed KNN algorithm when different dimensionality spaces were selected for each algorithm. The reasons could lie in that the KNN algorithm used was a simple vanilla KNN algorithm but HBL variant was benefiting from a weighted distance calculation modification.

3. Lastly, HBL algorithm outperformed ID3 algorithm.

7.3 Experiment Three : Comparison using Kappa Test

7.3.1 Introduction

In this experiment, we will compare HBL to other classifiers based on the level of agreement of each classifier to target output values. Instead of applying a paired student t-test, we will use Cohen's Kappa coefficient [11, 10] which demonstrates the level of agreement of the classifier to the actual data labels. A student paired t-test does not fully exploit all of the information from confusion matrices (e.g. true negatives and false positives). Kappa test uses all of the information in confusion matrices to find the level of agreement between classifiers and is mainly employed in diagnosis tasks where there are more than one diagnoses available [11].

We will run algorithms on full data sets using 10-fold cross validation and further apply statistical Ztest [81] to measure whether the obtained Cohen's coefficient is significant at 95% or not by bolding the classifier name if the results are significant.

Out of four different variants of back propagation which we have used in this thesis work, gradient descent is comparatively slower as it takes more time to converge[1]. Levenberg-Marquardt and Quasi Newton's method although very suitable for small problem size, are costly when network size (number of weights) becomes large [1]. With the above reasonings, we have decided the following subset of classifiers for this experiment: Resilient Back propagation (RPROP) algorithm from neural networks, HBL (HBL_3) from Hyperball learning algorithms along with KNN and ID3. These selected algorithms are shown in the Table 7.20.

Table 7.20: Third Experiment Algorithms

Selected Algorithms		
Algorithm	Category	Abbreviation Used
Resilient Back Propagation	Neural Networks	Rprop
HBL	Hyperball Learning	HBL
KNN	K Nearest Neighbours	KNN
ID3	Decision Trees	ID3

Experiment Details

The procedure for this experiment is given below in Algorithm 9.

Algorithm 9 Experiment 3: Kappa Test of Agreement

1: **for** each algorithm listed in Table 7.20 **do**
2: Train algorithm on each data set listed in Table 6.1 using 10-fold cross validation and for number of 30 runs.
Evaluate confusion matrix of each classifier.
3: **return** Confusion Matrix
4: **end for**

In above algorithm, 10-fold cross validation will be used to train each classifier for each data set in Table 7.20. Confusion matrices for each run are saved and averaged over all 30 runs. Kappa coefficient is calculated from confusion matrices. Kappa coefficient will show the level of agreement between classifier and actual classification labels and it is calculated as follows:

$$KAPPA = \frac{P(A) - P(E)}{1 - P(E)} \tag{7.1}$$

Where $P(A)$ is the relative agreement among classifier and $P(E)$ is the probability that agreement is due to chance. $P(E)$ is calculated as follows:

$$P(E) = \frac{\sum_{k=1}^{c}([\sum_{j=1}^{c}\sum_{i=1}^{m} f(i,k)c(i,j)].[\sum_{j=1}^{c}\sum_{i=1}^{m} f(i,j)c(j,k)])}{m^2} \tag{7.2}$$

$$P(A) = \frac{([\sum_{j=1}^{m}\sum_{j=1}^{c} f(i,j)c(i,j)])}{m} \tag{7.3}$$

Table 7.21: Kappa values Reference

Kappa Coeff. Reference Table	
≤ 0.20	Poor
0.21 - 0.40	Fair
0.41 - 0.60	Moderate
0.61 - 0.80	Good
0.81 - 1.00	Very good

In above equations, m denotes the number of samples, c number of classes. $f(i,j)$ denotes the actual probability of example i to be of class j. $c(i,j)$ can 0 or 1 values only. $c(i,j)$ is 1 iff j is the predicted class for i obtained from $p(i,j)$ which is the estimated probability of example i to be in class j and can be calculated as total number of samples in class j over total number of samples in all classes.

In order to test whether or not the obtained results are significant, Z Statistic test is used to check the significance of the obtained Kappa coefficient. Z Statistic test is calculated as follows:

$$Zscore = \frac{Kappa}{(Variance)^{\frac{1}{2}}} \tag{7.4}$$

In the above formula, *Variance* values are sample variances of respected kappa statistics. If Z score is greater than 1.65, then we can conclude that the classification results obtained from a classifier are significantly meaningful at 95% confidence level [12]. This has been shown in the tables provided by bolding the classifier name. On the Other hand, when classifier name is not balded in Tables in such tables, it means that the null hypothesis which states the results are not significant cannot be rejected.

Table 7.22: Kappa Score Distribution

Kappa Coeff.	Score
Insignificant	0
≤ 0.20	0
0.21 - 0.40	1
0.41 - 0.60	2
0.61 - 0.80	3
0.81 - 1.00	4

7.3.2 Results and Evaluation Methodology

Kappa test results are shown in Tables 7.23 to 7.31. For the sake of the reader's convenience, results showing confusion matrices are moved to Appendix B.

In order to do a comparison, we have assigned a score to each row of Table 7.21 as it is shown in Table 7.22. Using the above a score will be calculated for each algorithm considering all the data set results listed below. The comparison will be done in such a way that the algorithm with the highest score will be considered to have the highest level of agreement to actual output values, and will be known as a better algorithm. The obtained results are listed in Tables 7.23-7.31. In these tables Z Scores for each Kappa test is calculated in addition to Cohen's Kappa value.

Table 7.23: Iris: Kappa Test Results

Algorithm	Cohen's Kappa	Kappa Error	Confidence Interval	Z Score
RP	0.95	0.01	0.94-0.96	51.92
KNN	0.93	0.01	0.92-0.95	51.01
ID3	0.93	0.01	0.92-0.95	50.97
HBL	0.94	0.01	0.93-0.96	51.63

Table 7.24: Glass Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score
RP	0.58	0.1598	0.27-0.89	3.33
KNN	0.69	0.1225	0.45-0.93	5.29
ID3	0.58	0.1598	0.27-0.89	3.33
HBL	0.80	0.1072	0.59-1.00	5.97

Table 7.25: Zoo Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score
RP	0.87	0.1225	0.63-1.00	5.51
KNN	0.94	0.0840	0.78-1.00	6.05
ID3	0.85	0.1317	0.59-1.00	5.43
HBL	0.95	0.0772	0.80-1.00	6.11

Table 7.26: Wine Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score
RP	0.87	0.1022	0.66-1.00	5.10
KNN	0.80	0.1224	0.56-1.00	4.71
ID3	0.43	0.1742	0.09-0.77	2.52
HBL	0.86	0.1055	0.65-1.00	5.07

Table 7.27: Parkinson Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score
RP	0.49	0.3259	0.0-1.00	1.96
KNN	0.22	0.3756	0.0-0.96	0.94
ID3	0.17	0.3821	0.0-0.92	0.73
HBL	0.28	0.3760	0.0-1.00	1.00

Table 7.28: Pima Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score
RP	0.42	0.1156	0.20-0.65	3.44
KNN	0.34	0.1208	0.10-0.58	2.76
ID3	0.34	0.1207	0.10-0.57	2.74
HBL	0.40	0.1173	0.17-0.63	3.23

Table 7.29: Breast Cancer Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score
RP	0.92	0.0498	0.82-1.00	7.68
KNN	0.86	0.0640	0.74-0.99	7.21
ID3	0.83	0.0706	0.69-0.97	6.93
HBL	0.86	0.0635	0.74-0.99	7.22

Table 7.30: Connectionist Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score
RP	0.12	0.0616	0.00-0.24	1.82
KNN	0.42	0.0564	0.31-0.53	6.28
ID3	0.18	0.0578	0.02-0.22	1.48
HBL	0.44	0.0556	0.33-0.55	6.71

Table 7.31: Musk V2 Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score
RP	0.13	0.0656	0.00-0.10	0.81
KNN	0.01	0.0657	0.00-0.14	0.23
ID3	0.10	0.0593	0.00-0.20	2.33
HBL	0.15	0.0613	0.03-0.27	3.91

7.3.3 Results Discussion

In this part, the scoring of each algorithm is calculated based on the Kappa score distribution provided in Table (7.32) and it's presented in Table 7.32.

Table 7.32: Total Score Results

Algorithm	Total Score
RP	22
KNN	22
ID3	17
HBL	22

As it is shown in the score Table in 7.32, KNN, RP and HBL algorithms have equal score. All of these algorithms are superior to ID3 algorithm as ID3 has scored 17 which is the least score among all 4 algorithms. This methodology seems however to be unable to distinguish between RP, KNN and HBL algorithms as they have all earned an equal score of 22.

7.3.4 Comparison based on Cohen's Kappa Value

Another comparison can be using the exact value of Cohen's Kappa to decide which classifier has a better agreement to actual data output. For this comparison, if Classifier X has a higher Kappa value when it is compared to Classifier Y, then Classifier X has a better agreement to actual data labels and it is shown as $Classifier\ X > Classifier\ Y$. If result of a classifier is considered to be not significant(according to Tables 7.23 -7.31), then that comparison is ignored (Shown by $---$ in Table 7.33). Table 7.33 summarizes the results shown in Tables 7.23 to Table 7.31.

Table 7.33: Comparison Based on Exact Kappa Value

Summary Result	Data set
$RP > HBL > KNN > ID3$	Iris
$HBL > KNN > RP = ID3$	Glass
$HBL > KNN > RP > ID3$	Zoo
$RP > HBL > KNN > ID3$	Wine
$RP > ---$	Parkinson
$RP > KNN = HBL > ID3$	Pima
$RP > KNN = HBL > ID3$	Breast Cancer
$HBL > KNN > ---$	Connectionist
$HBL > ID3 > ---$	Musk V2

According to this table, shows that RP has a better agreement to actual data labels when it is compared to HBL algorithm (RP is better 4 times whereas HBL only 2 times). It can also be concluded that after RP variant, HBL algorithm has the best agreement as it has higher Kappa value in more number of cases when it is compared to KNN and ID3 algorithms.

7.4 Overall discussions

In this chapter we studied three different experiments to compare HBL algorithm to other classifiers. First experiment was comparing HBL algorithm to other available classifiers when 25% of sample size was selected and increased each time by 25%.

In this experiment first we performed an inter-HBL comparison to choose the best representative of HBL variants. The results obtained from this experiment showed that HBL outperforms other two variants of hyperball algorithm so HBL was chosen for further experiments. Next HBL was compared to BP algorithm variants. The obtained results from this comparison showed that HBL performed almost equal to most of BP variants. However, when sample size was considered GDM out performed HBL and when dimensionality considered, BFG had more number of wins beyond 50% level of dimensionality. Later on HBL was also compared to KNN and ID3 algorithms. HBL results outperformed ID3 at both small and large sample size selection. When HBL was compared to KNN algorithm, HBL showed better results. However, KNN algorithm employed for this thesis was basic vanilla algorithm.

In the second experiment, we compared HBL algorithm to other classifiers when the dimensionality of data (principal components) were increasing. Obtained results from this experiment showed that when dimensionality of data is changing ANN back propagation variants perform equally to the HBL algorithm.

Like the previous experiments HBL results outperformed both KNN and ID3 algorithm.

In last experiment HBL was compared to other classifiers according to Kappa[11] statistical measure. In this experiment using the Kappa Cohen's coefficient, we determined a level of agreement between the output

of each classifier and the actual data output. Further we evaluated significance of this obtained coefficient by can calculating a Z-score value using Z-statistical test. This Kappa value was also used to assign a weighted score to each of the classifiers for the comparison.

Obtained results for this experiment showed that HBL, KNN, and RP algorithms outperform ID3 classifier. However, the employed technique was unable to distinguish between HBL, KNN, and RP algorithms as they all score equally meaning that they all have the same level of agreement to data output labels in general.

Another comparison was done using the exact Kappa coefficient values. Summary table was generated and obtained results showed that RP algorithm outperforms HBL algorithm whereas HBL algorithm outperforms ID3 and KNN as it has a better agreement.

Chapter 8

Conclusion and Future Works

8.1 Conclusion

In this work we compared Hyperball Learning Algorithm to other classifiers based on increase on sample size, dimensionality increase and also based on the level of agreement. The following conclusions are made based on the data sets used in this work and the given data in Chapter 7 and Appendix B:

- HBL_3 with weighted sum distance calculation measure was the best HBL variant in the examined data sets.

- Considering the classification accuracy as measure and also full sample sizes can be deduced that HBL performs as good as ANN algorithms whereas HBL performs better than KNN and ID3 algorithm.

- When dimensionality of data increases and classification accuracy is the measure ANN algorithms and HBL algorithm perform indifferently. In the same context HBL algorithm is a better candidate when compared to KNN and ID3.

- From the last experiment and when Cohen's Kappa is a measure, it can be concluded that RP algorithm outperforms HBL and so has a higher level of agreement to actual target outputs. After RP, HBL algorithm performs better than ID3 and KNN algorithm.

8.2 Future Works

- A research can be done to determine ways to improve classifier comparisons.
- HBL can be further compared to other advanced KNN algorithms.
- Autonomous learning capabilities of HBL algorithm can be explored. HBL algorithm can then be compared to other existing autonomous learning algorithms.
- HBL algorithm was mainly designed to work as a parallel algorithm but in this thesis it was implemented as a sequential algorithm. HBL algorithms can be further implemented in parallel, and learning speed of HBL algorithm can be compared to other existing parallel algorithm.
- One of other features of HBL algorithm which was not employed in this thesis is related to its "learning from fallible expert" capability. HBL algorithm can evolve in a way that if it is miss supervised from a fallible data label, it can adapt its structure to correct the fallible information later on. This capability can further be examined and compared with similar algorithm if there exists.
- One of disadvantages of Instance Based Algorithms, specially KNN and HBL is that they are depending on all the data attributes to calculate the distance. HBL algorithm can be combined with method-

ologies like principal component analysis to use only the meaningful attributes in order to calculate a distance.

- In addition to above point, we can calculate the importance of each attribute using principal component analysis then while calculating distance between each two patterns bring in the importance of each attribute by adding a weight for each attribute we are using.

- Research can be done to decide the most suitable metric space for which HBL algorithm can have better accuracy such as Manhattan space.

- HBL algorithm centers a ball around each pattern. A modification to this methodology is to see if using an asymmetric geometric figure (with unequal radii) around the pattern could improve its accuracy.

- Another modification which can be tested on HBL algorithm is that instead of centering a ball around the pattern, we can center a ball around each dimension associated with data. In two dimensional space, this can be an ellipse which the pattern can be anywhere inside the ellipse.

- In HBL algorithm, a ball around a pattern starts with a maximum possible radius and later on as new balls are inserted into knowledge bank, this radius is shrunk if the ball is over lapping with any other balls belonging to a different category. However, there is no step involved in the algorithm to increase the radius of any balls if necessary. There can be situations in which the balls can grow their radius without overlapping other balls when the learning is done.

- And finally, HBL algorithm can be compared to more classifiers from Instance Based Learning category like: locally weighted regressions,

case-based reasoning, radial basis functions, etc.

- HBL algorithm can be further compared to Radial basis based methods, Gaussian Mixture methods and support vector machines so it can be further compared to these classifiers as well.

Bibliography

[1] A. Nigrin, *Neural networks for pattern recognition*. The MIT press, 1993, ISBN: 0-19-853864-2.

[2] T. Mitchell, *Machine Learning*. McGraw-Hill, 2004 ISBN: 0070428077.

[3] J. Rissanen, "A universal prior for integers and estimation by minimum description length," *The Annals of Statistics*, vol. 11, no. 2, pp. 416–431, 1983.

[4] D. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *SIAM Journal on Applied Mathematics*, vol. 11, pp. 431–441, 1963.

[5] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm.," tech. rep., 1993.

[6] R. Schoenberg, "Optimization with the quasi-newton method," *Unpublished manuscript, Aptech Systems, Maple Valley, WA*, 2001.

[7] V. Wojcik and B. Salami, "Machine Perception and Learning from the Evolutionary Viewpoint: The Hyperball Algorithms," 2008.

[8] H. Hotelling, "Analysis of a complex of statistical variables into principal components.," *Journal of Educational Psychology*, vol. 24, no. 6, pp. 417–441, 1933.

[9] C. Ferri, J. Hernández-Orallo, and R. Modroiu, "An experimental comparison of performance measures for classification," *Pattern Recognition Letters*, vol. 30, no. 1, pp. 27–38, 2009.

[10] C. Ferri, J. Hernández-Orallo, and R. Modroiu, "An experimental comparison of performance measures for classification," *Pattern Recognition Letters*, vol. 30, no. 1, pp. 27–38, 2009.

[11] D. Rossiter, "Technical Note: Statistical methods for accuracy assessment of classified thematic maps," tech. rep., 2004.

[12] R. Dwivedi, S. Kandrika, K. Ramana, *et al.*, "Comparison of classifiers of remote-sensing data for land-use/land-cover mapping," *Current Science*, vol. 86, no. 2, pp. 328–335, 2004.

[13] M. Hassoun, *Fundamentals of Artificial Neural Networks*. The MIT Press, 1995, ISBN: 026208239X.

[14] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. Wiley, New York, 2001,ISBN-10: 0471056693.

[15] V. Phansalkar and P. Sastry, "Analysis of the back-propagation algorithm with momentum," *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 505–506, 1994.

[16] S. Gallant, "Perceptron-based learning algorithms," *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 179–191, 1990.

[17] J. More, "The Levenberg-Marquardt algorithm: implementation and theory," *Numerical Analysis*, pp. 105–116, 1978.

[18] C. Kelley, *Solving nonlinear equations with Newton's method*. Society for Industrial Mathematics, 2003, ISBN: 0-89871-546-6.

[19] M. Poulton, *Computational neural networks for geophysical data processing*. Pergamon Pr, 2001, ISBN: 0080439861.

[20] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[21] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006.

[22] J. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[23] L. Breiman, *Classification and regression trees*. Chapman & Hall/CRC, 1984, ISBN: 9780412048418.

[24] D. Aha, D. Kibler, and M. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37–66, 1991.

[25] P. Zezula, *Similarity search: the metric space approach*. Springer-Verlag New York Inc, 2006, ISBN-10: 0387291466.

[26] M. Orr, "Introduction to radial basis function networks," tech. rep., 1996.

[27] A. Thompson, *Minkowski geometry.* Cambridge Univ Pr, 1996, ISBN:052140472X.

[28] J. Han and M. Kamber, *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2006, ISBN: 978-1-55860-901-3.

[29] S. Kullback and R. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[30] R. Miller and D. Siegmund, "Maximally selected chi square statistics," *Biometrics*, pp. 1011–1016, 1982.

[31] P. Cunningham and S. Delany, "k-Nearest neighbour classifiers," tech. rep., University College Dublin, 2007.

[32] Y. Rubner, C. Tomasi, and L. Guibas, "The earth mover's distance as a metric for image retrieval," *International Journal of Computer Vision*, vol. 40, no. 2, pp. 99–121, 2000.

[33] M. Li, X. Chen, X. Li, B. Ma, and P. Vitányi, "The similarity metric," *IEEE Transactions on Information Theory*, vol. 50, no. 12, pp. 3250–3264, 2004.

[34] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pp. 47–57, ACM, 1984.

[35] D. White and R. Jain, "Similarity indexing with the SS-tree," *Proceedings of the Twelfth International Conference on Data Engineering*, pp. 516–523, 1996.

[36] N. Katayama and S. Satoh, "The SR-tree: An index structure for high-dimensional nearest neighbor queries," in *Proceedings of the 1997 ACM SIGMOD International Conference on Management of data*, pp. 369–380, ACM, 1997.

[37] E. P. M. Lozano, J. Martinez, "A branch and bound algorithm for computing k-nearest neighbours,," *IEEE Transactions on computers.*, vol. 24, no. 3, pp. 743–750, 1975.

[38] M. Lenz, H. Burkhard, and S. Bruckner, "Applying case retrieval nets to diagnostic tasks in technical domains," *Advances in Case-Based Reasoning*, pp. 219–233, 1996.

[39] M. Lenz and H. Burkhard, "Case retrieval nets: Basic ideas and extensions," *KI-96: Advances in Artificial Intelligence*, pp. 227–239, 1996.

[40] B. Smyt and E. McKenna, "Footprint-based retrieval," *In Proceedings of the Third International Conference on Case-Based Reasoning*, pp. 343–357, 1999.

[41] J. Schaaf, "Fish and Shrink. A next step towards efficient case retrieval in large scaled case bases," *Advances in Case-Based Reasoning*, pp. 362–376, 1996.

[42] S. Hernandez Rodriguez, J. Martinez Trinidad, and J. Carrasco Ochoa, "Fast k most similar neighbor classifier for mixed data (tree k-msn)," *Pattern Recognition Archive*, vol. 43, pp. 873–886, March 2010.

[43] R. E. Pakelska, *The Dissimilarity Representation for Pattern recognition*, vol. 43. 2005, ISBN: 978-981-270-317-0(ebook).

[44] D. Y. H. Chang, "Relaxation metric adaptation metric adaptation and its application to semi-supervised clustering and content-based image retrieval," *Pattern Recognition Papers*, vol. 39, no. 3, pp. 1905–1917, 2006.

[45] E. P. M. Lozano, J. Martinez, "Experimental study on prototype optimization algorithms for prototype based classification in vector spaces," *Pattern Recognition Papers*, vol. 39, no. 3, pp. 1827–1838, 2006.

[46] S. Bandyopadhyay and U. Maulik, "Efficient prototype reordering in nearest neighbor classification," *Pattern Recognition*, vol. 35, no. 12, pp. 2791–2799, 2002.

[47] L. J.H Friedman, F. Baskett, "An algorithm for finding nearest neighbours," *IEEE Transactions on computers.*, vol. 24, pp. 100–1006, 1975.

[48] P. Grother, G. Candela, and J. Blue, "Fast implementations of nearest neighbor classifiers," *Pattern Recognition*, vol. 30, no. 3, pp. 459–465, 1997.

[49] Y. Chen, Y. Hung, T. Yen, and C. Fuh, "Fast and versatile algorithm for nearest neighbor search based on a lower bound tree," *Pattern Recognition*, vol. 40, no. 2, pp. 360–375, 2007.

[50] E. Lee and S. Chae, "Fast design of reduced complexity nearest-neighbor classifiers using triangular inequality," *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 20, no. 5, pp. 562–566.

[51] J. Oncina, F. Thollard, E. Gómez-Ballester, L. Micó, and F. Moreno-Seco, "A Tabular Pruning Rule in Tree-Based Fast Nearest Neighbor Search Algorithms," *Pattern Recognition and Image Analysis*, pp. 306–313, 2007.

[52] J. Lai, Y. Liaw, and J. Liu, "Fast k-nearest-neighbor search based on projection and triangular inequality," *Pattern Recognition*, vol. 40, no. 2, pp. 351–359, 2007.

[53] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 891–923, 1998.

[54] F. Moreno-Seco, L. Micó, and J. Oncina, "Approximate nearest neighbour search with the fukunaga and narendra algorithm and its application to chromosome classification," *Progress in Pattern Recognition, Speech and Image Analysis*, pp. 322–328, 2003.

[55] D. Mladenić, "Feature subset selection in text-learning," *Machine Learning: ECML-98*, pp. 95–100, 1998.

[56] R. Kohavi and G. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[57] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," *Data Mining and Knowledge Discovery*, vol. 6, no. 2, pp. 153–172, 2002.

[58] S. Delany and P. Cunningham, "An analysis of case-base editing in a spam filtering system," *7 th European Conference in Case-Based Reasoning*, vol. 7, pp. 3–25, 2004.

[59] P. Hart, "The condensed nearest neighbor rule (Corresp.)," *IEEE Transactions on Information Theory*, vol. 14, no. 3, pp. 515–516, 1968.

[60] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour, "An algorithm for a selective nearest neighbor decision rule," *IEEE Transactions on Information Theory*, vol. 21(6), pp. 665–669, 1975.

[61] D. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 2, no. 3, pp. 408–421, 1972.

[62] I. Tomek, "An experiment with the edited nearest-neighbor rule," *Systems, Man and Cybernetics, IEEE Transactions on*, pp. 448–452, 1976.

[63] A. Frank and A. Asuncion, "UCI machine learning repository." `http://archive.ics.uci.edu/ml/`, 2010.

[64] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *International Joint Conference on Artificial Intelligence*, vol. 14, pp. 1137–1145, 1995.

[65] P. Burman, "A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods," *Biometrika*, pp. 503–514, 1989.

[66] J. Banks, B. Nelson, and D. Nicol, *Discrete-Event System Simulation*. Prentice Hall, 2009, ISBN: 0130887021.

[67] S. Robson, M. Oliveira, E. Agropecuária, and M. Salete, *Data transformation for privacy-preserving data mining*. PhD thesis, 2005.

[68] S. Kotsiantis and P. Pintelas, "Logitboost of simple bayesian classifier," *Informatica*, vol. 29, no. 1, pp. 53–59, 2005.

[69] J. Dy and C. Brodley, "Feature selection for unsupervised learning," *The Journal of Machine Learning Research*, vol. 5, p. 889, 2004.

[70] J. Smith, J. Everhart, W. Dickson, W. Knowler, and R. Johannes, "Using the ADAP learning algorithm to forecast the onset of diabetes mellitus," in *Proceedings of the Annual Symposium on Computer Application in Medical Care*, pp. 261–265, American Medical Informatics Association, 1988.

[71] S. Aeberhard, D. Coomans, and O. De Vel, "Comparison of classifiers in high dimensional settings," *Dept. Math. Statist., James Cook Univ., North Queensland, Australia, Tech. Rep*, 1992.

[72] M. Little, P. McSharry, E. Hunter, J. Spielman, and L. Ramig, "Suitability of dysphonia measurements for telemonitoring of Parkinsons disease," *Nature Publishing Group*, 2008.

[73] O. Mangasarian and W. Wolberg, "Cancer diagnosis via linear programming," *SIAM News*, vol. 23, no. 5, pp. 1–18, 1990.

[74] W. Wolberg and O. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 87, no. 23, pp. 9193–9196, 1990.

[75] O. Mangasarian, R. Setiono, and W. Wolberg, "Pattern recognition via linear programming: Theory and application to medical diagnosis," *Pattern Recognition*, pp. 22–30, 1990.

[76] B. Yegnanarayana, *Artificial neural networks*. PHI Learning Pvt. Ltd., 2004, ISBN=8120312538.

[77] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 3, pp. 21–26, Washington, 1990.

[78] M. Leshno and Y. Spector, "The effect of training data set size and the complexity of the separation function on neural network classification capability: The two-group case," *Naval Research Logistics*, vol. 44, no. 8, pp. 699–717, 1997.

[79] Y. Kim, "Comparison of the decision tree, artificial neural network, and linear regression methods based on the number and types of independent variables and sample size," *Expert Systems with Applications*, vol. 34, no. 2, pp. 1227–1234, 2008.

[80] M. Sordo and Q. Zeng, "On sample size and classification accuracy: A performance comparison," *Biological and Medical Data Analysis*, pp. 193–201, 2005.

[81] B. Agarwal, *Basic statistics*. New Age International, 2006, ISBN: 8122418147.

[82] S. Dudani, "The distance-weighted k-nearest-neighbor rule," *SMC*, vol. 6, pp. 325–327, April 1976.

[83] F. Plastria, S. De Bruyne, and E. Carrizosa, "Dimensionality Reduction for Classification?," in *Advanced Data Mining and Applications: 4th International Conference, ADMA 2008, Chengdu, China, October 8-10*, pp. 411–417, Springer-Verlag New York Inc, 2008.

[84] R. Raich, J. Costa, and A. Hero III, "On dimensionality reduction for classification and its application," University of Michigan.

[85] S. Mosci, L. Rosasco, and A. Verri, "Dimensionality reduction and generalization," in *Proceedings of the 24th International Conference on Machine Learning*, p. 664, ACM, 2007.

[86] A. Hyvarinen and J. Karhunen, "Independent Component Analysis," 2001.

[87] S. Vempala, *The random projection method*. Amer Mathematical Society, 2005, ISBN: 0821837931.

[88] J. Shlens, "A tutorial on principal component analysis," tech. rep., Center for Neural Science, New York University New York City, NY 10003-6603 and Systems Neurobiology Laboratory, Salk Institute for Biological Studies La Jolla, CA 92037, 2009.

Appendices

Appendix A

Error Back Propagation Algorithm

BP algorithm is used to train a multi layer feed forward network using gradient descent to approximate an unknown function, based on some training data consisting of pairs (x,t), where the vector x represents input patterns to the network, and the vector t represents the corresponding target (desired output). Recall from mathematics, the overall gradient with respect to the entire training set is the sum of the gradients for each pattern; in what follows we will therefore describe how to compute the gradient for a single training pattern. Lets select node j from output layer, in this case weights from previous layer i to node j will be denoted as w_{ji}. For node j, we can write the following equations:

$$e_j(n) = d_j(n) - y_j(n) \tag{A.1}$$

Where $e_j(n)$ is the error at output node j. n refers to iteration n in the training process. $y_j(n)$ is the actual output and $d_j(n)$ is the desired output.

$$E(n) = \frac{1}{2} \sum_{j \in C_{out}} e_j^2(n) \tag{A.2}$$

Where $E(n)$ is the instantaneous value of error energy and C_{out} is set of all nodes in output layer.

$$net_j(n) = \sum w_{ji} y_i(n) \tag{A.3}$$

$$y_j(n) = f(net_j(n)) \tag{A.4}$$

Where $f()$ is the activation function for neuron j. We are interested in calculating the $\Delta w_{ji}(n)$ (the change in weight required for w_{ji}). $\Delta w_{ji}(n)$ which is applied to w_{ji} is proportional to $\frac{\partial E(n)}{\partial w_{ji}(n)}$. We apply the chain rule of differentiation to simplify the above expression:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n}{\partial e_j(n)} . \frac{\partial e_j(n)}{\partial y_j(n)} . \frac{\partial y_j(n)}{\partial net_j(n)} . \frac{\partial net_j(n)}{\partial w_{ji}(n)}$$

Where $= \frac{\partial E(n}{\partial e_j(n)} = e_j(n)$ according to A.2, $\frac{\partial y_j(n)}{\partial net_j(n)} = f'(net_j(n))$ according to A.4, $\frac{\partial e_j(n)}{\partial y_j(n)} = -1$ according to A.1, and $\frac{\partial net_j(n)}{\partial w_{ji}(n)} = y_i(n)$ according to A.3. So our expression will be simplified as:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n) \quad f'(net_j(n)) \quad y_i(n) \tag{A.5}$$

$\Delta w_{ji}(n)$ will be applied to $w_{ji}(n)$, where:

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)} = \eta e_j(n) f'(net_j(n)) y_i(n) \tag{A.6}$$

$$= \eta . \delta_j(n) . y_i(n) \tag{A.7}$$

In above expression, η is called the learning rate which is a value $0 < \eta < 1$. δ_j is derivative of error with respect to net_j, or the local gradient. Local gradient δ_j can be calculated as follows:

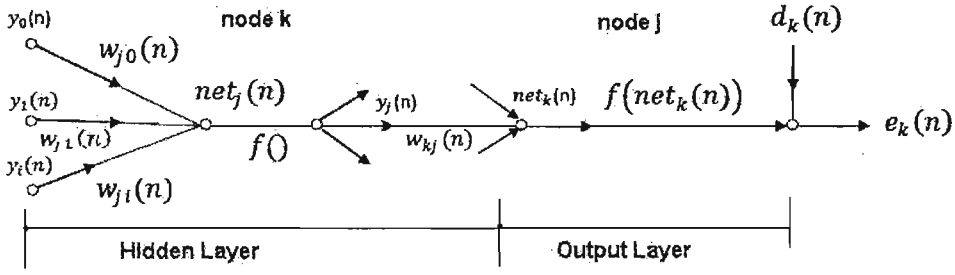

Figure A.1: Signal Flow Diagram from Hidden Layer to Output layer

$$\delta_j = -\frac{\partial E(n)}{\partial net_j(n)} \tag{A.8}$$

$$= -\frac{\partial E(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial net_j(n)} \tag{A.9}$$

$$= e_j(n) f'(net_j(n)) \tag{A.10}$$

In above expression we had assumed that node j belongs to output layer and we have shown that δ_j can be calculated based on the derived formula above. the second case can be when node j belongs to hidden layer.

In figure A.1, signal flow from hidden layer to output layer is shown. In this figure node j belongs to hidden layer and node k belongs to output layer. for this figure, local gradient will be calculated as follows:

$$\delta_j = -\frac{\partial E(n)}{\partial net_j(n)}$$

$$= -\frac{\partial E(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial net_j(n)}$$

$$= -\frac{\partial E(n)}{\partial y_j(n)} \cdot f'(net_j(n)) \tag{A.11}$$

$E(n)$ in this case will be as follows:

$$E(n) = \frac{1}{2} \sum_{k \in output_nodes} e_k^2(n) \tag{A.12}$$

In this case, expression $\frac{\partial E(n)}{\partial y_j(n)}$ should be calculated.

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n) \cdot \frac{\partial e_k(n)}{\partial y_j(n)}$$

$$= \sum_k e_k(n) \frac{\partial e_k(n)}{\partial net_k(n)} \frac{net_k(n)}{\partial y_j(n)} \tag{A.13}$$

Given that $e_k(n) = d_k(n) - y_k(n)$ we can write

$$e_k(n) = d_k(n) - f(net_k(n)) \tag{A.14}$$

so:

$$\frac{\partial e_k(n)}{\partial net_k(n)} = -f'(net_k(n)) \tag{A.15}$$

for node k, we can write:

$$net_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n) \tag{A.16}$$

so:

$$\frac{\partial net_k(n)}{\partial y_j(n)} = w_{kj}(n) \tag{A.17}$$

Now equation A.13 can be re-written as:

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k e_k(n) f'(net_k(n)) w_{kj}(n)$$
$$= -\sum_k \delta_k(n) w_{kj}(n)$$

Now if we apply that on δ_j equation in A.11, we'll have:

$$\delta_j(n) = f'_j(net_j(n)).\sum_k \delta_k(n) w_{kj}(n) \tag{A.18}$$

So as we have shown above, local gradient of a hidden node is expressed in terms of local gradient of the output neuron. Recall that $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$ we can propagate the error backwards to calculate $\Delta w_{ji}(n)$ for each weight connection.

So in order to calculate the error for unit j, we must first know the error of all its posterior nodes. Again, as long as there are no cycles in the network, there is an ordering of nodes from the output back to the input that respects this condition. For example, we can simply use the reverse of the order in which activity was propagated forward.

Vanilla gradient descent usually works well for simple models, but as the error space becomes too complex vanilla gradient descent method takes a long time to converge: The reason why is that the problem is *stiff* in the sense that the few places where small step sizes are required ruins it for the whole problem. For example when descending the walls of a very steep bowl, very small steps should be taken to avoid "rattling out" of the bowl. On the other hand, when moving along the gently sloping parts, longer steps are required. The problem is addressed by choosing a step that is some constant times the negative gradient rather than a step of constant length in direction of the negative gradient. This is equivalent to moving slowly in shallow regions and moving quickly in steep regions. Another issue is that curvature of the error surface may not be the same

in all directions. This will cause different components of the gradient in different directions to have different value which might slow down the algorithm [4].

Appendix B

Experiment Results

We have included detailed obtained results in this Appendix. The columns titles are explained as follows:

Table B.1: Results Title Description

Column Title	Description
Sample size	Selected percentage of sample size
Min	Minimum observed classification accuracy value
Max	Maximum observed classification accuracy value
Mean	Average of all 30 runs of classification accuracy values
Std	Standard Deviation
Dimension	Portion of top principal components used

B.1 Experiment one: Sample Size Effect

Table B.2: Glass Dataset Results

Algorithm	Sample Size	Min	Max	Mean	Std
LM	25%	31.87	62.00	47.76	5.29
	50%	45.94	62.73	53.87	3.75
	75%	54.43	64.59	59.14	2.38
	100%	57.09	67.78	63.24	2.30
RP	25%	40.86	57.19	49.79	4.53
	50%	49.48	66.59	56.77	3.63
	75%	53.35	63.26	58.35	2.29
	100%	58.07	66.39	64.77	2.24
bfg	25%	53.36	67.45	60.85	3.18
	50%	51.59	66.52	59.35	3.27
	75%	49.43	61.85	56.43	2.75
	100%	59.06	67.78	64.67	1.95
gdm	25%	50.48	69.69	61.90	4.74
	50%	45.05	58.30	52.90	3.16
	75%	55.74	64.18	60.76	2.52
	100%	56.44	66.86	60.69	2.58
KNN	25%	39.16	62.97	53.83	5.12
	50%	55.73	69.70	65.03	3.74
	75%	61.67	73.68	68.09	2.57
	100%	68.62	76.42	72.47	2.13
ID3	25%	38.21	61.63	50.19	5.05
	50%	46.9	62.6	55.11	3.69
	75%	57.81	66.51	61.28	2.55
	100%	61.35	70.32	66.04	2.15
HBL_1	25%	45.54	59.49	50.94	2.39
	50%	56.61	64.31	61.68	1.72
	75%	66.49	72.29	68.75	1.51
	100%	68.41	75.16	71.47	1.43
HBL_2	25%	50.94	58.49	55.72	2.65
	50%	61.68	68.22	64.54	1.62
	75%	68.75	75.62	72.16	1.48
	100%	70.09	77.10	73.47	1.40
HBL_3	25%	52.68	66.71	58.49	2.83
	50%	63.63	73.19	68.22	1.98
	75%	73.27	78.24	75.62	1.72
	100%	72.58	79.60	77.10	1.49

Table B.3: Cohens d effect size for Glass

Algorithm	Size	GDM	BFG	RP	KNN	ID3	HBL_1	HBL_2	HBL_3
LM	25%	0.82	0.83	NS	0.50	NS	0.69	0.68	0.78
	50%	NS	0.61	0.37	0.81	NS	0.88	0.91	0.93
	75%	0.31	0.47	NS	0.95	0.38	0.96	0.96	0.97
	100%	0.46	0.35	0.33	0.93	0.54	0.94	0.94	0.97
gdm	25%	–	NS	0.79	0.63	0.79	0.63	0.62	0.39
	50%	–	0.71	0.49	0.85	0.40	0.92	0.35	0.67
	75%	–	0.63	0.45	0.94	NS	0.94	0.94	0.96
	100%	–	NS	NS	0.94	0.75	0.95	0.95	0.97
BFG	25%	–	–	0.82	0.64	0.81	0.66	0.65	0.35
	50%	–	–	0.35	0.59	0.44	0.71	0.58	0.83
	75%	–	–	0.35	0.96	0.66	0.96	0.96	0.97
	100%	–	–	NS	0.95	0.73	0.96	0.96	0.98
RP	25%	–	–	–	0.39	NS	0.62	0.62	0.75
	50%	–	–	–	0.72	NS	0.81	0.91	0.94
	75%	–	–	–	0.96	0.50	0.96	0.96	0.98
	100%	–	–	–	0.95	0.71	0.95	0.95	0.97
KNN	25%	–	–	–	–	0.40	NS	NS	0.48
	50%	–	–	–	–	0.75	NS	0.81	0.88
	75%	–	–	–	–	0.93	NS	NS	0.69
	100%	–	–	–	–	0.87	0.27	NS	0.84
ID3	25%	–	–	–	–	–	0.62	0.61	0.74
	50%	–	–	–	–	–	0.83	0.89	0.93
	75%	–	–	–	–	–	0.94	0.94	0.96
	100%	–	–	–	–	–	0.90	0.89	0.95
HBL_1	25%	–	–	–	–	–	–	NS	0.43
	50%	–	–	–	–	–	–	0.89	0.94
	75%	–	–	–	–	–	–	NS	0.76
	100%	–	–	–	–	–	–	NS	0.80
HBL_2	25%	–	–	–	–	–	–	–	0.42
	50%	–	–	–	–	–	–	–	0.94
	75%	–	–	–	–	–	–	–	0.76
	100%	–	–	–	–	–	–	–	0.79

Table B.4: Iris Dataset Results

Algorithm	Sample Size	Min	Max	Mean	Std
LM	25%	79.17	94.67	89.01	3.83
	50%	91.96	100.00	98.11	2.36
	75%	92.80	97.50	95.61	1.09
	100%	95.33	98.67	96.87	0.95
RP	25%	90.00	97.50	95.19	2.08
	50%	95.66	100.00	97.53	1.12
	75%	94.27	98.33	96.73	1.25
	100%	94.00	98.00	96.47	0.91
bfg	25%	88.83	100.00	93.61	2.74
	50%	94.64	97.50	96.75	0.87
	75%	93.85	98.26	96.67	1.17
	100%	94.67	98.00	96.58	0.83
gdm	25%	88.33	100.00	95.27	3.34
	50%	88.83	98.75	94.38	2.63
	75%	92.35	98.33	95.86	1.77
	100%	92.03	99.73	95.25	1.82
KNN	25%	89.71	96.97	93.72	1.65
	50%	93.64	99.91	97.10	1.29
	75%	92.36	96.38	94.81	1.10
	100%	93.64	97.19	95.29	0.82
ID3	25%	86.17	97.86	91.18	3.34
	50%	90.85	95.38	93.02	1.27
	75%	90.39	94.86	92.49	1.02
	100%	92.67	97.54	95.33	1.17
HBL_1	25%	91.89	94.59	92.52	1.16
	50%	94.67	97.33	95.82	0.84
	75%	91.96	95.54	93.87	0.69
	100%	92.00	96.00	93.02	0.82
HBL_2	25%	90.15	94.46	92.53	1.24
	50%	94.16	96.75	95.81	0.63
	75%	92.28	95.76	93.85	0.71
	100%	91.59	94.36	93.01	0.71
HBL_3	25%	91.35	97.64	94.59	1.33
	50%	95.59	99.40	97.33	0.96
	75%	93.16	97.91	95.54	0.93
	100%	94.30	98.03	96.00	1.00

Table B.5: Cohens d effect size for Iris

Algorithm	Size	GDM	BFG	RP	KNN	ID3	HBL_1	HBL_2	HBL_3
LM	25%	0.66	0.57	0.71	0.62	0.29	0.53	0.53	0.70
	50%	0.60	0.36	NS	0.26	0.80	0.54	0.78	0.83
	75%	NS	0.42	0.43	0.34	0.83	0.69	0.69	NS
	100%	0.49	NS	NS	0.66	0.59	0.91	0.92	0.41
gdm	25%	–	0.26	NS	0.28	0.52	0.48	0.48	NS
	50%	–	0.52	0.62	0.55	0.31	0.35	NS	0.39
	75%	–	0.26	0.27	0.34	0.76	0.60	0.60	NS
	100%	–	0.42	0.39	NS	NS	0.62	0.63	0.25
BFG	25%	–	–	0.31	NS	0.37	0.25	0.25	NS
	50%	–	–	0.36	NS	0.86	0.48	0.49	0.67
	75%	–	–	NS	0.63	0.89	0.82	0.82	0.47
	100%	–	–	NS	0.61	0.52	0.91	0.92	0.30
RP	25%	–	–	–	0.36	0.58	0.62	0.61	NS
	50%	–	–	–	NS	0.88	0.65	NS	0.55
	75%	–	–	–	0.63	0.88	0.82	0.82	0.48
	100%	–	–	–	0.56	0.48	0.89	0.90	NS
KNN	25%	–	–	–	–	0.43	0.39	0.38	0.28
	50%	–	–	–	–	0.85	0.51	0.64	0.80
	75%	–	–	–	–	0.74	0.46	0.46	0.34
	100%	–	–	–	–	NS	0.81	0.83	0.36
ID3	25%	–	–	–	–	–	0.26	0.26	0.56
	50%	–	–	–	–	–	0.79	0.69	0.78
	75%	–	–	–	–	–	0.62	0.61	0.84
	100%	–	–	–	–	–	0.75	0.77	0.29
HBL_1	25%	–	–	–	–	–	–	NS	0.64
	50%	–	–	–	–	–	–	0.87	0.91
	75%	–	–	–	–	–	–	NS	0.71
	100%	–	–	–	–	–	–	NS	0.85
HBL_2	25%	–	–	–	–	–	–	–	0.63
	50%	–	–	–	–	–	–	–	0.91
	75%	–	–	–	–	–	–	–	0.71
	100%	–	–	–	–	–	–	–	0.86

Table B.6: Zoo Dataset Results

Algorithm	Sample Size	Min	Max	Mean	Std
LM	25%	50.37	70.20	60.15	5.75
	50%	82.95	96.00	89.67	2.72
	75%	84.13	92.56	88.34	2.09
	100%	83.85	95.98	91.30	2.40
RP	25%	57.28	75.81	68.14	4.63
	50%	76.29	91.07	84.15	3.26
	75%	80.10	92.72	87.15	2.91
	100%	85.14	94.17	90.14	2.04
bfg	25%	59.83	85.17	77.38	5.60
	50%	80.33	92.41	87.25	2.94
	75%	86.80	95.75	91.93	2.04
	100%	89.91	95.09	92.80	1.44
gdm	25%	61.52	79.89	68.61	4.54
	50%	78.00	90.86	83.14	2.90
	75%	83.53	93.99	89.30	2.91
	100%	84.10	96.27	91.48	2.29
KNN	25%	82.14	98.40	88.89	3.93
	50%	88.29	98.60	96.16	2.77
	75%	91.89	97.81	96.30	1.90
	100%	93.75	97.47	96.06	1.54
ID3	25%	63.91	76.91	71.25	2.84
	50%	71.98	80.61	76.19	2.09
	75%	82.75	89.15	85.12	1.49
	100%	87.00	90.80	88.88	0.89
HBL_1	25%	82.86	94.13	87.34	2.62
	50%	87.23	97.22	92.02	2.47
	75%	90.23	98.27	94.66	2.07
	100%	89.16	95.74	93.06	1.56
HBL_2	25%	81.27	94.05	88.12	2.72
	50%	90.45	99.71	95.60	2.23
	75%	92.80	98.18	97.22	2.00
	100%	91.74	99.42	95.74	1.76
HBL_3	25%	84.16	96.63	90.33	3.40
	50%	90.95	97.89	96.00	1.37
	75%	93.84	98.91	97.34	1.59
	100%	94.15	98.00	97.01	1.74

Table B.7: Cohens d effect size for Zoo

Algorithm	Size	GDM	BFG	RP	KNN	ID3	HBL_1	HBL_2	HBL_3
LM	25%	0.63	0.84	0.61	0.95	0.77	0.95	0.95	0.95
	50%	0.76	0.39	0.68	0.76	0.94	0.41	0.97	0.97
	75%	NS	0.66	NS	0.89	0.66	0.83	0.91	0.92
	100%	NS	0.35	NS	0.76	0.55	0.40	0.73	0.81
gdm	25%	–	0.65	NS	0.92	0.33	0.93	0.93	0.94
	50%	–	0.57	NS	0.92	0.81	0.85	0.97	0.97
	75%	–	0.46	0.35	0.82	0.67	0.73	0.85	0.86
	100%	–	0.33	0.30	0.76	0.60	0.37	0.72	0.81
BFG	25%	–	–	0.67	0.77	0.57	0.75	0.77	0.81
	50%	–	–	0.45	0.84	0.91	0.66	0.91	0.91
	75%	–	–	0.69	0.74	0.89	0.55	0.79	0.83
	100%	–	–	0.60	0.74	0.85	NS	0.67	0.80
RP	25%	–	–	–	0.92	0.38	0.93	0.93	0.94
	50%	–	–	–	0.89	0.82	0.81	0.97	0.97
	75%	–	–	–	0.88	0.40	0.83	0.90	0.91
	100%	–	–	–	0.85	0.37	0.63	0.83	0.88
KNN	25%	–	–	–	–	0.93	NS	NS	NS
	50%	–	–	–	–	0.97	0.62	0.72	0.74
	75%	–	–	–	–	0.96	0.38	NS	0.29
	100%	–	–	–	–	0.94	0.70	NS	0.28
ID3	25%	–	–	–	–	–	0.95	0.95	0.95
	50%	–	–	–	–	–	0.96	0.98	0.98
	75%	–	–	–	–	–	0.94	0.96	0.97
	100%	–	–	–	–	–	0.85	0.93	0.95
HBL_1	25%	–	–	–	–	–	–	NS	0.44
	50%	–	–	–	–	–	–	0.86	0.87
	75%	–	–	–	–	–	–	0.53	0.59
	100%	–	–	–	–	–	–	0.63	0.77
HBL_2	25%	–	–	–	–	–	–	–	0.34
	50%	–	–	–	–	–	–	–	0.84
	75%	–	–	–	–	–	–	–	NS
	100%	–	–	–	–	–	–	–	0.34

Table B.8: Wine Dataset Results

Algorithm	Sample Size	Min	Max	Mean	Std
LM	25%	80.17	96.00	87.94	4.41
	50%	81.41	92.22	87.55	2.38
	75%	87.20	95.48	90.89	1.99
	100%	85.45	95.58	92.51	1.83
RP	25%	70.50	85.00	78.38	4.36
	50%	80.56	92.31	88.55	2.43
	75%	90.08	94.29	92.41	1.24
	100%	91.04	95.00	93.15	1.01
bfg	25%	85.00	98.00	92.15	3.13
	50%	85.47	94.17	90.83	2.05
	75%	86.46	93.41	90.68	1.62
	100%	89.85	94.40	92.55	1.18
gdm	25%	78.67	96.67	86.92	4.77
	50%	82.69	96.89	89.97	3.17
	75%	88.30	94.78	92.75	1.47
	100%	88.41	95.03	93.08	1.49
KNN	25%	78.16	86.77	81.93	2.23
	50%	79.85	89.32	83.98	2.30
	75%	82.30	88.99	86.23	1.47
	100%	85.09	88.23	86.56	0.87
ID3	25%	47.60	63.55	55.46	3.18
	50%	54.02	60.67	57.13	2.00
	75%	56.14	63.15	59.69	1.61
	100%	60.05	64.39	62.19	0.93
HBL_1	25%	77.53	86.52	82.51	2.22
	50%	68.18	79.55	76.59	2.40
	75%	84.21	89.47	87.47	1.40
	100%	86.52	90.45	88.41	1.10
HBL_2	25%	71.96	80.63	76.59	2.19
	50%	79.64	86.50	82.51	1.74
	75%	84.40	90.28	87.46	1.75
	100%	85.99	90.45	88.42	1.13
HBL_3	25%	75.87	85.66	79.53	2.17
	50%	82.23	90.33	86.51	1.97
	75%	86.47	93.67	89.47	1.79
	100%	87.51	93.29	90.46	1.27

Table B.9: Cohens d effect size for Wine

Algorithm	Size	GDM	BFG	RP	KNN	ID3	HBL_1	HBL_2	HBL_3
LM	25%	NS	0.48	0.74	0.65	0.97	0.61	0.85	0.77
	50%	0.40	0.59	NS	0.61	0.99	0.92	0.63	NS
	75%	0.47	NS	0.42	0.80	0.99	0.70	0.67	0.35
	100%	NS	NS	NS	0.90	1.00	0.81	0.80	0.55
gdm	25%	–	0.54	0.68	0.56	0.97	0.51	0.81	0.71
	50%	–	NS	0.24	0.73	0.99	0.92	0.52	NS
	75%	–	0.56	NS	0.91	1.00	0.88	0.85	0.71
	100%	–	NS	NS	0.94	1.00	0.87	0.87	0.69
BFG	25%	–	–	0.88	0.88	0.99	0.87	0.94	0.92
	50%	–	–	0.45	0.84	0.99	0.95	0.89	0.73
	75%	–	–	0.52	0.82	0.99	0.73	0.69	0.33
	100%	–	–	NS	0.94	1.00	0.88	0.87	0.65
RP	25%	–	–	–	0.46	0.95	0.51	NS	NS
	50%	–	–	–	0.69	0.99	0.93	0.53	0.77
	75%	–	–	–	0.92	1.00	0.88	0.85	0.69
	100%	–	–	–	0.96	1.00	0.91	0.91	0.76
KNN	25%	–	–	–	–	0.98	NS	0.77	0.48
	50%	–	–	–	–	0.99	0.84	NS	0.74
	75%	–	–	–	–	0.99	0.40	0.36	0.70
	100%	–	–	–	–	1.00	0.68	0.68	0.87
ID3	25%	–	–	–	–	–	0.98	0.97	0.98
	50%	–	–	–	–	–	0.98	0.98	0.99
	75%	–	–	–	–	–	0.99	0.99	0.99
	100%	–	–	–	–	–	1.00	1.00	1.00
HBL_1	25%	–	–	–	–	–	–	0.80	0.56
	50%	–	–	–	–	–	–	NS	0.69
	75%	–	–	–	–	–	–	NS	0.53
	100%	–	–	–	–	–	–	NS	0.65
HBL_2	25%	–	–	–	–	–	–	–	0.56
	50%	–	–	–	–	–	–	–	0.92
	75%	–	–	–	–	–	–	–	0.49
	100%	–	–	–	–	–	–	–	0.65

Table B.10: BreastCancer Dataset Results

Algorithm	Sample Size	Min	Max	Mean	Std
LM	25%	75.17	86.65	81.41	3.30
	50%	82.08	91.45	87.62	2.36
	75%	88.83	100.62	93.72	2.50
	100%	93.10	98.79	96.30	1.40
RP	25%	77.26	87.21	80.66	2.30
	50%	79.95	94.92	88.87	3.50
	75%	82.57	91.55	86.91	2.17
	100%	91.49	98.73	94.88	1.82
bfg	25%	73.07	93.74	82.81	5.18
	50%	82.22	98.38	88.21	3.62
	75%	89.76	99.63	94.87	2.47
	100%	88.68	97.17	95.59	2.53
gdm	25%	73.29	94.38	86.24	5.15
	50%	84.75	95.96	90.65	2.96
	75%	88.75	99.61	94.59	2.75
	100%	90.18	97.90	96.50	2.41
KNN	25%	75.47	86.88	81.87	2.39
	50%	81.63	91.55	86.34	2.21
	75%	85.49	93.25	89.06	1.77
	100%	90.64	95.86	93.77	1.13
HBL_1	25%	77.00	88.54	83.33	3.34
	50%	82.51	89.92	87.01	1.90
	75%	87.01	93.04	89.46	1.52
	100%	89.31	94.52	92.24	1.10
HBL_2	25%	76.93	90.99	83.56	3.26
	50%	84.33	91.80	88.72	2.03
	75%	84.05	92.69	88.72	1.90
	100%	85.99	90.09	87.69	1.08
HBL_3	25%	78.60	87.98	83.83	2.80
	50%	84.93	92.55	88.64	1.93
	75%	85.58	92.84	89.12	1.80
	100%	91.48	95.71	93.83	0.93

Algorithm	Size	GDM	BFG	RP	KNN	ID3	HBL_1	HBL_2	HBL_3
LM	25%	0.49	NS	NS	NS	0.86	0.28	0.31	0.37
	50%	0.49	NS	NS	0.27	0.85	NS	0.80	0.80
	75%	NS	NS	0.82	0.73	0.86	0.72	0.75	0.73
	100%	NS	NS	0.40	0.71	0.93	0.85	0.96	0.72
gdm	25%	–	0.32	0.57	0.48	0.87	0.32	0.30	0.28
	50%	–	0.35	0.26	0.64	0.90	0.59	0.30	0.30
	75%	–	NS	0.84	0.77	0.87	0.76	0.78	0.76
	100%	–	NS	0.36	0.59	0.88	0.75	0.92	0.59
BFG	25%	–	–	0.26	NS	0.81	NS	NS	NS
	50%	–	–	NS	0.30	0.80	NS	0.60	0.60
	75%	–	–	0.86	0.80	0.89	0.80	0.81	0.80
	100%	–	–	NS	0.42	0.84	0.65	0.90	0.42
RP	25%	–	–	–	NS	0.88	0.42	0.46	0.53
	50%	–	–	–	0.40	0.83	0.31	0.88	0.88
	75%	–	–	–	0.48	NS	0.56	0.41	0.48
	100%	–	–	–	0.34	0.87	0.66	0.92	0.34
KNN	25%	–	–	–	–	0.90	NS	0.28	0.35
	50%	–	–	–	–	0.81	NS	0.84	0.84
	75%	–	–	–	–	0.59	NS	NS	NS
	100%	–	–	–	–	0.88	0.57	0.94	NS
ID3	25%	–	–	–	–	–	0.89	0.90	0.91
	50%	–	–	–	–	–	0.86	0.96	0.96
	75%	–	–	–	–	–	0.67	0.53	0.60
	100%	–	–	–	–	–	0.78	0.57	0.90
HBL_1	25%	–	–	–	–	–	–	NS	NS
	50%	–	–	–	–	–	–	0.70	0.70
	75%	–	–	–	–	–	–	NS	NS
	100%	–	–	–	–	–	–	0.90	0.62
HBL_2	25%	–	–	–	–	–	–	–	NS
	50%	–	–	–	–	–	–	–	0.69
	75%	–	–	–	–	–	–	–	NS
	100%	–	–	–	–	–	–	–	0.95

Table B.11: Parkinson Dataset Results

Algorithm	Sample Size	Min	Max	Mean	Std
LM	25%	70.00	86.00	78.07	4.27
	50%	79.44	89.89	82.86	2.72
	75%	84.84	91.76	88.24	1.80
	100%	82.54	90.33	86.35	1.63
RP	25%	77.50	89.50	84.08	3.06
	50%	84.72	91.09	87.16	1.41
	75%	80.76	88.24	84.54	1.96
	100%	83.63	89.84	86.91	1.47
bfg	25%	66.50	86.50	80.36	4.51
	50%	78.44	87.89	84.33	2.12
	75%	85.51	90.33	87.73	1.21
	100%	83.47	89.32	86.21	1.50
gdm	25%	78.17	88.33	83.49	2.59
	50%	79.78	85.56	82.77	1.53
	75%	81.57	88.38	84.80	1.56
	100%	83.47	88.82	85.71	1.18
KNN	25%	87.82	99.14	93.69	2.89
	50%	83.11	96.04	89.27	3.41
	75%	87.85	94.81	91.05	1.71
	100%	78.94	85.31	82.02	1.62
ID3	25%	65.59	73.92	70.01	2.15
	50%	69.94	77.62	74.15	2.22
	75%	74.52	82.87	78.56	1.93
	100%	77.31	82.76	80.50	1.14
HBL_1	25%	79.49	89.72	85.40	2.98
	50%	82.91	91.10	87.62	2.03
	75%	87.15	92.69	89.72	1.25
	100%	75.34	80.02	77.44	1.28
HBL_2	25%	87.24	100.43	91.65	3.67
	50%	83.48	95.01	89.24	2.48
	75%	87.53	93.46	90.34	1.73
	100%	81.28	85.14	83.02	0.84
HBL_3	25%	87.36	99.68	91.82	3.24
	50%	83.44	95.01	89.90	2.60
	75%	86.24	94.91	91.78	1.87
	100%	81.31	86.45	84.26	1.25

Table B.12: Cohens d effect size for Parkinson

Algorithm	Size	GDM	BFG	RP	KNN	ID3	HBL_1	HBL_2	HBL_3
LM	25%	0.61	NS	0.63	0.91	0.77	0.71	0.86	0.88
	50%	NS	0.29	0.71	0.72	0.87	0.70	0.85	0.86
	75%	0.71	NS	0.70	0.62	0.93	0.43	0.51	0.69
	100%	NS	NS	NS	0.80	0.90	0.95	0.79	0.59
gdm	25%	–	0.39	NS	0.88	0.94	0.32	0.79	0.82
	50%	–	0.39	0.83	0.78	0.91	0.80	0.75	0.78
	75%	–	0.72	NS	0.89	0.87	0.87	0.86	0.90
	100%	–	NS	0.41	0.79	0.91	0.96	0.80	0.51
BFG	25%	–	–	0.44	0.87	0.83	0.55	0.81	0.82
	50%	–	–	0.62	0.66	0.92	0.62	0.77	0.79
	75%	–	–	0.70	0.75	0.94	0.63	0.66	0.79
	100%	–	–	0.23	0.80	0.91	0.95	0.80	0.58
RP	25%	–	–	–	0.85	0.94	NS	0.75	0.78
	50%	–	–	–	0.37	0.96	NS	0.68	0.72
	75%	–	–	–	0.87	0.84	0.84	0.84	0.88
	100%	–	–	–	0.84	0.93	0.96	0.85	0.70
KNN	25%	–	–	–	–	0.98	0.82	0.29	0.29
	50%	–	–	–	–	0.93	0.28	0.64	0.57
	75%	–	–	–	–	0.96	0.40	NS	NS
	100%	–	–	–	–	0.48	0.84	0.36	0.61
ID3	25%	–	–	–	–	–	0.95	0.96	0.97
	50%	–	–	–	–	–	0.95	0.97	0.97
	75%	–	–	–	–	–	0.96	0.96	0.96
	100%	–	–	–	–	–	0.78	0.78	0.84
HBL_1	25%	–	–	–	–	–	–	0.68	0.72
	50%	–	–	–	–	–	–	0.57	0.63
	75%	–	–	–	–	–	–	NS	0.54
	100%	–	–	–	–	–	–	0.93	0.94
HBL_2	25%	–	–	–	–	–	–	–	NS
	50%	–	–	–	–	–	–	–	0.27
	75%	–	–	–	–	–	–	–	0.37
	100%	–	–	–	–	–	–	–	0.50

Table B.13: Madelon Dataset Results

Algorithm	Sample Size	Min	Max	Mean	Std
RP	25%	51.53	56.94	53.99	1.37
	50%	55.22	59.84	57.38	1.17
	75%	52.97	58.87	55.12	1.49
	100%	53.96	59.42	56.73	1.26
gdm	25%	48.27	55.95	52.19	2.01
	50%	52.76	58.74	55.22	1.56
	75%	53.37	59.09	56.02	1.48
	100%	53.91	58.60	56.46	1.23
KNN	25%	61.32	74.44	67.76	3.33
	50%	62.20	74.17	68.78	2.65
	75%	66.68	73.90	70.21	2.01
	100%	68.58	73.63	70.85	1.31
HBL_2	25%	62.48	76.95	69.54	3.13
	50%	64.14	75.25	70.00	2.70
	75%	66.85	75.94	71.09	2.24
	100%	68.82	74.64	71.88	1.55

Table B.14: Cohens Effect Size for Madelon

Algorithm	Size	GDM	RP	KNN	HBL_2
GDM	25%	–	0.46	0.94	0.96
	50%	–	0.62	0.95	0.96
	75%	–	0.29	0.97	0.97
	100%	–	NS	0.98	0.98
RP	25%	–	–	0.94	0.95
	50%	–	–	0.94	0.95
	75%	–	–	0.97	0.97
	100%	–	–	0.98	0.98
KNN	25%	–	–	–	0.27
	50%	–	–	–	NS
	75%	–	–	–	NS
	100%	–	–	–	0.34

Table B.15: Pima Dataset Results

Algorithm	Sample Size	Min	Max	Mean	Std
RP	25%	67.98	75.78	71.33	1.72
	50%	71.34	77.08	74.75	1.24
	75%	74.48	76.95	75.37	0.68
	100%	68.69	80.22	73.74	2.59
gdm	25%	72.14	78.16	74.51	1.27
	50%	73.96	77.78	75.72	0.96
	75%	74.87	77.74	76.34	0.68
	100%	73.81	81.80	78.27	1.78
KNN	25%	64.47	74.05	68.70	2.33
	50%	67.08	72.56	70.09	1.43
	75%	66.66	73.64	70.65	1.97
	100%	68.94	75.05	72.02	1.55
ID3	25%	59.38	70.91	65.31	2.76
	50%	65.71	76.35	69.71	2.30
	75%	65.95	72.95	70.16	1.74
	100%	68.57	73.20	71.69	1.15
HBL_1	25%	56.75	68.04	61.88	3.14
	50%	60.21	68.75	64.31	2.06
	75%	60.10	65.19	62.40	1.43
	100%	65.19	69.05	66.89	1.04
HBL_2	25%	58.09	73.14	66.31	3.60
	50%	61.77	71.03	67.07	2.11
	75%	66.31	72.95	69.81	1.39
	100%	68.18	73.84	71.29	1.22
HBL_3	25%	63.35	73.79	69.31	2.66
	50%	65.43	73.86	69.85	1.88
	75%	69.65	74.78	72.34	1.47
	100%	71.84	77.68	74.48	1.35

Table B.16: cohens d EffectSize for Pima

Algorithm	Size	RP	KNN	ID3	HBL_1	HBL_2	HBL_3
gdm	25%	0.72	0.84	0.91	0.93	0.84	0.78
	50%	0.40	0.92	0.86	0.96	0.91	0.82
	75%	0.58	0.89	0.92	0.99	0.95	0.87
	100%	0.71	0.88	0.91	0.97	0.92	0.77
RP	25%	–	0.54	0.80	0.88	0.67	0.41
	50%	–	0.87	0.81	0.95	0.74	0.38
	75%	–	0.85	0.89	0.99	0.93	0.80
	100%	–	0.37	0.46	0.87	0.52	NS
KNN	25%	–	–	0.55	0.78	0.37	NS
	50%	–	–	NS	0.85	0.34	NS
	75%	–	–	NS	0.92	NS	0.44
	100%	–	–	NS	0.89	NS	0.65
ID3	25%	–	–	–	0.50	NS	0.59
	50%	–	–	–	0.78	0.34	0.69
	75%	–	–	–	0.93	NS	0.56
	100%	–	–	–	0.91	NS	0.74
HBL_1	25%	–	–	–	–	0.55	0.79
	50%	–	–	–	–	0.70	0.84
	75%	–	–	–	–	0.93	0.96
	100%	–	–	–	–	0.89	0.95
HBL_2	25%	–	–	–	–	–	0.43
	50%	–	–	–	–	–	0.52
	75%	–	–	–	–	–	0.66
	100%	–	–	–	–	–	0.78

Table B.17: Musk Dataset Results

Algorithm	Sample Size	Min	Max	Mean	Std
RP	25%	52.01	70.11	61.77	3.73
	50%	60.07	71.28	66.68	2.97
	75%	64.18	74.39	70.59	2.47
	100%	71.83	77.00	74.71	1.50
gdm	25%	50.66	71.27	60.06	4.66
	50%	61.28	74.08	67.30	3.41
	75%	67.21	76.62	71.85	2.35
	100%	70.37	80.53	75.54	2.32
KNN	25%	59.82	69.94	64.56	2.28
	50%	64.77	74.29	68.42	2.08
	75%	67.59	75.22	70.63	1.77
	100%	72.79	78.73	75.53	1.43
ID3	25%	45.65	64.28	56.97	3.90
	50%	54.88	60.93	58.38	1.85
	75%	57.55	64.04	61.47	1.51
	100%	63.61	67.92	65.73	1.09
HBL_1	25%	56.80	71.11	63.36	3.58
	50%	59.32	71.25	66.04	2.91
	75%	65.56	72.78	69.58	1.68
	100%	71.13	76.58	73.47	1.37
HBL_2	25%	58.47	71.43	65.26	3.49
	50%	62.31	71.47	67.08	2.02
	75%	68.53	74.29	71.33	1.67
	100%	73.56	77.97	76.12	1.29
HBL_3	25%	61.74	74.83	67.75	3.12
	50%	65.81	75.07	70.60	2.33
	75%	69.33	78.08	73.65	1.97
	100%	76.09	79.72	77.49	0.92

Table B.18: Cohens d Effect size for Musk

Algorithm	Size	RP	KNN	ID3	HBL_1	HBL_2	HBL_3
Gdm	25%	NS	0.52	0.34	0.37	0.53	0.70
	50%	NS	NS	0.85	NS	0.70	0.82
	75%	NS	0.28	0.93	0.49	NS	0.38
	100%	NS	NS	0.94	0.48	NS	0.48
RP	25%	–	0.41	0.53	NS	0.43	0.66
	50%	–	0.32	0.86	NS	0.66	0.82
	75%	–	NS	0.91	NS	NS	0.57
	100%	–	0.27	0.96	0.40	0.45	0.75
KNN	25%	–	–	0.76	NS	NS	0.50
	50%	–	–	0.93	0.43	0.50	0.80
	75%	–	–	0.94	0.29	NS	0.63
	100%	–	–	0.97	0.59	NS	0.63
ID3	25%	–	–	–	0.65	0.75	0.84
	50%	–	–	–	0.84	0.85	0.90
	75%	–	–	–	0.93	0.95	0.96
	100%	–	–	–	0.95	0.97	0.99
HBL_1	25%	–	–	–	–	NS	0.55
	50%	–	–	–	–	0.54	0.77
	75%	–	–	–	–	0.46	0.74
	100%	–	–	–	–	0.71	0.87
HBL_2	25%	–	–	–	–	–	0.35
	50%	–	–	–	–	–	0.67
	75%	–	–	–	–	–	0.54
	100%	–	–	–	–	–	0.52

Table B.19: Connectionist Dataset Results

Algorithm	Sample Size	Min	Max	Mean	Std
RP	25%	71.20	78.26	74.21	1.74
	50%	79.42	86.82	82.78	1.66
	75%	83.53	88.59	86.59	1.08
	100%	55.97	63.75	59.81	1.87
gdm	25%	56.96	65.05	61.09	1.82
	50%	57.18	63.40	61.31	1.24
	75%	58.59	64.04	61.23	1.42
	100%	56.03	66.26	60.72	2.68
KNN	25%	49.02	61.08	55.19	3.53
	50%	49.58	58.22	54.39	2.17
	75%	54.83	61.09	58.10	1.74
	100%	53.18	59.10	56.74	1.61
ID3	25%	41.48	54.43	47.32	2.97
	50%	45.92	55.93	49.84	2.04
	75%	47.23	55.52	50.07	2.11
	100%	49.15	52.84	51.41	0.90
HBL_1	25%	45.93	59.70	52.81	2.88
	50%	47.99	57.74	52.03	2.23
	75%	50.75	56.66	54.41	1.71
	100%	51.24	56.05	53.73	1.22
HBL_2	25%	49.06	64.43	58.60	3.45
	50%	49.67	60.09	55.96	2.95
	75%	54.14	60.86	57.35	1.76
	100%	56.19	60.16	58.20	1.05
HBL_3	25%	57.50	66.83	62.26	2.54
	50%	53.95	64.37	58.09	2.66
	75%	56.33	65.56	60.34	2.02
	100%	55.65	61.55	59.30	1.25

Table B.20: Cohens d Size Effect for Connectionist Data Set

Algorithm	Size	RP	KNN	ID3	HBL_1	HBL_2	HBL_3
gdm	25%	0.96	0.72	0.94	0.86	0.41	0.26
	50%	0.99	0.89	0.96	0.93	0.72	0.55
	75%	1.00	0.70	0.95	0.91	0.77	NS
	100%	NS	0.67	0.92	0.86	0.53	0.32
RP	25%	–	0.96	0.98	0.98	0.94	0.94
	50%	–	0.99	0.99	0.99	0.97	0.96
	75%	–	0.99	1.00	1.00	1.00	0.99
	100%	–	0.66	0.94	0.89	0.47	NS
KNN	25%	–	–	0.77	0.35	0.44	0.75
	50%	–	–	0.73	0.47	NS	0.42
	75%	–	–	0.90	0.73	NS	0.51
	100%	–	–	0.90	0.73	0.47	0.66
ID3	25%	–	–	–	0.68	0.87	0.94
	50%	–	–	–	0.46	0.83	0.89
	75%	–	–	–	0.75	0.88	0.93
	100%	–	–	–	0.73	0.96	0.96
HBL_1	25%	–	–	–	–	0.67	0.87
	50%	–	–	–	–	0.48	0.69
	75%	–	–	–	–	0.65	0.85
	100%	–	–	–	–	0.89	0.91
HBL_2	25%	–	–	–	–	–	0.52
	50%	–	–	–	–	–	NS
	75%	–	–	–	–	–	0.62
	100%	–	–	–	–	–	0.43

B.2 Experiment Two: Data Dimensionality Effect

Table B.21: Glass Dataset Results

Algorithm	Dimension	Min	Max	Mean	Std
RP	25%	56.14	62.54	60.10	1.41
	50%	58.41	68.28	63.00	1.76
	75%	58.45	65.95	62.19	2.16
	100%	58.07	66.39	64.77	2.24
BFG	25%	56.54	64.98	60.98	1.80
	50%	59.35	69.03	63.06	2.41
	75%	58.08	69.58	61.99	2.69
	100%	59.06	67.78	64.67	1.95
KNN	25%	58.84	63.94	61.36	1.23
	50%	61.76	69.66	66.79	1.81
	75%	61.95	70.69	65.55	2.37
	100%	69.80	76.48	72.67	1.49
ID3	25%	51.39	60.90	55.44	2.14
	50%	56.21	66.59	61.17	3.03
	75%	52.86	68.23	59.29	3.78
	100%	61.35	70.32	66.10	2.16
HBL_3	25%	61.87	64.81	63.09	0.92
	50%	69.10	77.06	72.20	1.72
	75%	70.27	80.41	75.95	2.62
	100%	74.51	79.81	77.09	1.34

Table B.22: Cohen d Effect Size Table for Glass

Algorithm	Dimension	RP	KNN	ID3	HBL_3
BFG	25%	NS	NS	0.81	0.59
	50%	NS	0.65	0.32	0.95
	75%	NS	0.57	0.38	0.93
	100%	NS	0.95	0.73	0.98
RP	25%	–	0.43	0.79	0.78
	50%	–	0.73	0.34	0.97
	75%	–	0.59	0.42	0.94
	100%	–	0.95	0.71	0.97
KNN	25%	–	–	0.86	0.62
	50%	–	–	0.75	0.96
	75%	–	–	0.70	0.90
	100%	–	–	0.87	0.84
ID3	25%	–	–	–	0.92
	50%	–	–	–	0.97
	75%	–	–	–	0.93
	100%	–	–	–	0.95

Table B.23: Zoo Dataset Results

Algorithm	Dimension	Min	Max	Mean	Std
RP	25%	84.16	93.53	89.15	2.25
	50%	83.41	92.95	89.07	2.21
	75%	86.88	94.36	90.54	1.83
	100%	85.14	94.17	90.14	2.04
BFG	25%	89.96	94.98	92.29	1.32
	50%	88.16	94.13	90.67	1.30
	75%	87.27	95.98	92.14	1.92
	100%	89.91	95.09	92.80	1.44
KNN	25%	85.82	95.12	91.22	2.29
	50%	89.24	98.98	94.49	2.62
	75%	89.01	97.65	92.66	1.85
	100%	93.75	98.15	96.06	1.54
ID3	25%	79.27	83.87	81.50	1.25
	50%	81.48	87.52	84.75	1.67
	75%	79.93	90.31	85.78	2.43
	100%	87.00	90.80	88.88	0.89
HBL_3	25%	90.35	98.71	93.72	1.74
	50%	87.93	95.23	92.12	1.81
	75%	92.37	98.30	95.45	1.41
	100%	94.15	98.14	97.01	1.74

Table B.24: Cohen d Effect Size Table for Zoo

Algorithm	Dimension	RP	KNN	ID3	HBL_3
BFG	25%	0.64	0.28	0.97	0.42
	50%	0.40	0.68	0.89	NS
	75%	0.39	NS	0.82	0.70
	100%	0.60	0.74	0.85	0.80
RP	25%	–	0.41	0.90	0.75
	50%	–	0.74	0.74	0.58
	75%	–	0.50	0.74	0.83
	100%	–	0.85	0.37	0.88
KNN	25%	–	–	0.93	0.52
	50%	–	–	0.91	NS
	75%	–	–	0.85	0.65
	100%	–	–	0.94	0.28
ID3	25%	–	–	–	0.97
	50%	–	–	–	0.96
	75%	–	–	–	0.93
	100%	–	–	–	0.95

Table B.25: Wine Dataset Results

Algorithm	Dimension	Min	Max	Mean	Std
RP	25%	80.95	88.72	84.57	1.58
	50%	84.08	91.62	88.07	1.90
	75%	82.61	91.59	88.02	1.90
	100%	91.04	95.00	93.15	1.01
BFG	25%	80.88	87.61	84.54	1.68
	50%	85.00	91.63	87.81	1.76
	75%	84.87	91.60	88.51	1.67
	100%	89.85	94.40	92.55	1.18
KNN	25%	78.13	84.02	80.75	1.33
	50%	80.29	88.03	84.16	1.92
	75%	80.78	91.82	85.24	2.43
	100%	85.09	88.23	86.56	0.87
ID3	25%	55.04	61.22	58.25	1.88
	50%	55.40	64.94	59.52	2.10
	75%	55.91	65.86	61.12	2.32
	100%	60.05	64.39	62.19	0.93
HBL_3	25%	80.77	86.14	83.68	1.30
	50%	79.43	89.26	85.26	2.40
	75%	81.32	90.66	85.50	2.59
	100%	87.51	93.29	90.46	1.27

Table B.26: Cohen d Effect Size Table for Wine

Algorithm	Dimension	RP	KNN	ID3	HBL_3
BFG	25%	0.01	0.78	0.99	0.27
	50%	0.07	0.70	0.99	0.17
	75%	0.13	0.61	0.99	0.57
	100%	0.27	0.94	1.00	0.65
RP	25%	–	NS	0.99	0.29
	50%	–	NS	0.99	NS
	75%	–	NS	0.99	0.48
	100%	–	NS	1.00	0.76
KNN	25%	–	–	0.99	0.74
	50%	–	–	0.99	0.76
	75%	–	–	0.98	NS
	100%	–	–	1.00	0.87
ID3	25%	–	–	–	0.99
	50%	–	–	–	0.99
	75%	–	–	–	0.98
	100%	–	–	–	1.00

Table B.27: Parkinson Dataset Results

Algorithm	Dimension	Min	Max	Mean	Std
RP	25%	84.08	90.24	86.77	1.37
	50%	69.79	74.20	72.74	0.78
	75%	84.55	93.29	88.43	1.50
	100%	83.63	89.84	86.91	1.47
BFG	25%	80.93	90.20	85.48	1.95
	50%	83.55	92.39	89.08	1.90
	75%	83.53	91.29	88.20	2.09
	100%	83.47	89.32	86.21	1.50
KNN	25%	75.72	82.11	79.43	1.34
	50%	81.12	83.56	82.32	0.68
	75%	78.99	84.94	81.63	1.35
	100%	78.94	85.31	82.02	1.62
ID3	25%	72.69	80.46	75.92	2.29
	50%	73.55	82.88	78.04	2.54
	75%	76.46	86.31	80.40	2.48
	100%	77.31	82.76	80.50	1.14
HBL_3	25%	77.23	83.20	80.60	1.63
	50%	81.26	84.14	82.86	0.78
	75%	79.87	87.17	83.86	1.76
	100%	81.31	86.45	84.26	1.25

Table B.28: Cohen d Effect Size Table for Parkinson

Algorithm	Dimension	RP	KNN	ID3	HBL_3
BFG	25%	0.35	0.87	0.91	0.80
	50%	0.98	0.92	0.93	0.66
	75%	NS	0.88	0.86	0.74
	100%	0.23	0.80	0.91	0.58
RP	25%	–	0.94	0.94	0.90
	50%	–	0.99	0.82	0.87
	75%	–	0.92	0.89	0.81
	100%	–	0.84	0.93	0.70
KNN	25%	–	–	0.68	0.37
	50%	–	–	0.75	0.84
	75%	–	–	0.29	0.58
	100%	–	–	0.48	0.61
ID3	25%	–	–	–	0.76
	50%	–	–	–	0.90
	75%	–	–	–	0.63
	100%	–	–	–	0.84

Table B.35: Breast Dataset Results

Algorithm	Dimension	Min	Max	Mean	Std
RP	25%	86.79	94.88	90.54	1.77
	50%	91.37	99.25	94.75	2.25
	75%	89.22	95.49	92.24	1.76
	100%	91.49	98.73	94.88	1.82
KNN	25%	85.90	93.44	89.37	1.82
	50%	86.88	96.42	91.05	2.11
	75%	89.35	95.64	92.52	1.90
	100%	90.64	95.86	93.77	1.13
ID3	25%	79.42	89.25	84.13	2.97
	50%	82.74	89.74	86.57	1.72
	75%	83.17	94.27	89.26	2.22
	100%	87.17	91.63	89.31	1.26
HBL_3	25%	85.20	92.06	88.83	1.80
	50%	87.98	97.16	92.10	2.42
	75%	88.93	95.79	92.17	1.82
	100%	91.48	95.71	93.83	0.93

Table B.29: Pima Dataset Results

Algorithm	Dimension	Min	Max	Mean	Std
RP	25%	70.68	73.56	72.16	0.75
	50%	69.79	74.20	72.74	0.78
	75%	73.69	77.33	75.34	0.90
	100%	68.69	80.22	73.74	2.59
BFG	25%	70.96	73.44	72.32	0.63
	50%	70.97	74.61	72.86	0.88
	75%	71.48	76.82	75.11	1.03
	100%	73.28	82.54	78.04	2.22
KNN	25%	63.48	66.39	65.17	0.68
	50%	68.42	71.24	69.64	0.80
	75%	69.58	72.58	71.03	0.78
	100%	68.94	75.05	72.02	1.55
ID3	25%	64.88	67.41	65.99	0.73
	50%	65.69	69.98	68.17	0.96
	75%	65.75	72.83	69.07	1.71
	100%	68.57	73.20	71.69	1.15
HBL_3	25%	66.30	68.88	67.46	0.56
	50%	68.84	71.97	70.59	0.81
	75%	71.49	76.16	73.32	1.00
	100%	71.84	77.68	74.48	1.35

Table B.36: Cohen d Effect Size Table for Breast

Algorithm	Dimension	RP	KNN	ID3	HBL_3
RP	25%	–	0.31	0.80	0.43
	50%	–	0.65	0.90	0.34
	75%	–	NS	0.60	NS
	100%	–	0.34	0.87	0.34
KNN	25%	–	–	0.73	NS
	50%	–	–	0.76	0.54
	75%	–	–	0.62	NS
	100%	–	–	0.88	NS
ID3	25%	–	–	–	0.69
	50%	–	–	–	0.83
	75%	–	–	–	0.58
	100%	–	–	–	0.90

Table B.30: Cohen d Effect Size Table for Pima

Algorithm	Dimension	RP	KNN	ID3	HBL_3
BFG	25%	NS	0.98	0.98	0.97
	50%	NS	0.89	0.93	0.76
	75%	NS	0.91	0.91	0.66
	100%	0.67	0.84	0.87	0.70
RP	25%	–	0.98	0.97	0.96
	50%	–	0.89	0.93	0.71
	75%	–	0.93	0.92	0.72
	100%	–	0.37	0.46	NS
KNN	25%	–	–	0.50	0.88
	50%	–	–	0.64	0.96
	75%	–	–	0.59	0.79
	100%	–	–	NS	0.65
ID3	25%	–	–	–	0.75
	50%	–	–	–	0.95
	75%	–	–	–	0.84
	100%	–	–	–	0.74

Table B.37: Connectionist Dataset Results

Algorithm	Dimension	Min	Max	Mean	Std
RP	25%	34.34	39.39	36.84	1.42
	50%	48.48	57.07	53.25	2.01
	75%	57.27	64.24	60.71	1.75
	100%	55.97	63.75	59.81	1.87
BFG	25%	44.44	48.28	46.62	1.05
	50%	60.40	67.37	63.55	1.67
	75%	67.07	74.34	70.27	1.92
	100%	56.03	66.26	60.72	2.68
KNN	25%	30.90	36.57	33.72	1.18
	50%	46.56	53.37	49.88	1.69
	75%	55.16	62.04	58.08	1.67
	100%	53.18	59.10	56.74	1.61
ID3	25%	32.96	37.00	35.00	1.00
	50%	41.38	49.90	46.04	1.81
	75%	43.41	54.38	49.65	2.79
	100%	49.15	52.84	51.41	0.90
HBL_3	25%	45.43	49.09	47.13	1.02
	50%	49.70	55.21	52.46	1.46
	75%	58.25	63.94	61.18	1.59
	100%	55.65	61.55	59.30	1.25

Table B.31: Iris Dataset Results

Algorithm	Dimension	Min	Max	Mean	Std
RP	25%	57.33	68.67	63.44	2.66
	50%	56.00	64.67	60.71	2.03
	75%	71.33	79.33	75.42	2.07
	100%	94.00	98.00	96.47	0.91
BFG	25%	58.00	66.67	63.58	2.05
	50%	56.00	68.00	62.18	3.18
	75%	68.00	80.00	74.40	3.02
	100%	94.67	98.00	96.58	0.83
KNN	25%	53.68	65.09	59.28	2.69
	50%	56.80	65.52	61.84	1.97
	75%	70.70	80.98	76.54	2.29
	100%	93.64	97.19	95.29	0.82
ID3	25%	56.90	70.79	65.02	3.22
	50%	58.29	65.41	61.51	2.01
	75%	70.33	79.61	75.96	2.64
	100%	92.67	97.54	95.33	1.17
HBL_3	25%	56.44	68.61	62.05	2.90
	50%	55.16	65.28	60.36	2.26
	75%	71.01	79.43	75.66	2.42
	100%	94.30	98.03	96.00	1.00

Table B.38: Cohen d Effect Size Table for Connectionist

Algorithm	Dimension	RP	KNN	ID3	HBL_3
BFG	25%	0.97	0.99	0.98	0.24
	50%	0.94	0.97	0.98	0.92
	75%	0.93	0.96	0.97	0.93
	100%	NS	0.67	0.92	0.32
RP	25%	–	0.76	0.59	0.97
	50%	–	0.67	0.88	0.98
	75%	–	0.61	0.92	NS
	100%	–	0.66	0.94	NS
KNN	25%	–	–	0.51	0.99
	50%	–	–	0.74	0.99
	75%	–	–	0.88	0.69
	100%	–	–	0.90	0.66
ID3	25%	–	–	–	0.99
	50%	–	–	–	0.99
	75%	–	–	–	0.93
	100%	–	–	–	0.96

Table B.32: Cohen d Effect Size Table for Iris

Algorithm	Dimension	RP	KNN	ID3	HBL_3
BFG	25%	NS	0.67	NS	0.29
	50%	NS	NS	NS	0.60
	75%	NS	0.37	NS	NS
	100%	NS	0.61	0.52	0.30
RP	25%	–	0.61	0.26	NS
	50%	–	0.27	NS	0.53
	75%	–	NS	NS	NS
	100%	–	0.56	0.48	NS
KNN	25%	–	–	0.70	0.44
	50%	–	–	NS	NS
	75%	–	–	NS	NS
	100%	–	–	NS	0.36
ID3	25%	–	–	–	0.44
	50%	–	–	–	0.64
	75%	–	–	–	NS
	100%	–	–	–	0.29

Table B.33: Madelon Dataset Results

Algorithm	Dimension	Min	Max	Mean	Std
RP	25%	55.38	59.12	57.35	0.94
	50%	53.73	57.04	55.37	0.96
	75%	53.96	56.92	55.57	0.65
	100%	52.63	58.02	54.92	1.24
KNN	25%	66.50	71.18	68.59	1.11
	50%	66.68	70.48	68.44	0.92
	75%	68.14	71.29	69.70	0.82
	100%	68.58	73.63	70.85	1.31
ID3	25%	41.12	44.78	43.57	0.88
	50%	45.62	50.23	48.17	1.05
	75%	47.92	51.19	49.30	0.80
	100%	46.14	52.07	49.80	1.53
HBL_3	25%	64.25	69.62	67.51	1.29
	50%	66.57	72.37	69.23	1.29
	75%	67.16	72.24	69.38	1.24
	100%	68.50	75.24	72.23	1.53

Table B.34: Cohen d Effect Size Table for Madelon

Algorithm	Dimension	RP	KNN	ID3	HBL_3
RP	25%	–	0.98	0.99	0.98
	50%	–	0.99	0.96	0.98
	75%	–	0.99	0.97	0.99
	100%	–	0.99	0.88	0.99
KNN	25%	–	–	1.00	0.41
	50%	–	–	1.00	NS
	75%	–	–	1.00	NS
	100%	–	–	0.99	0.44
ID3	25%	–	–	–	1.00
	50%	–	–	–	1.00
	75%	–	–	–	0.99
	100%	–	–	–	0.99

Table B.39: Musk V2 Dataset Results

Algorithm	Dimension	Min	Max	Mean	Std
RP	25%	39.12	54.55	47.61	3.74
	50%	62.60	68.80	65.78	1.81
	75%	68.09	75.12	71.73	1.58
	100%	72.16	78.44	75.53	1.71
BFG	25%	47.92	56.77	51.91	2.43
	50%	56.37	67.03	62.16	2.76
	75%	67.28	76.93	73.66	2.17
	100%	72.21	78.61	75.82	1.56
KNN	25%	44.44	57.73	50.97	3.13
	50%	59.66	67.21	63.08	1.97
	75%	71.29	76.48	73.41	1.28
	100%	72.79	78.73	75.53	1.43
ID3	25%	36.16	51.80	44.24	4.38
	50%	52.06	63.96	58.03	2.64
	75%	59.77	70.58	65.13	2.50
	100%	63.61	67.92	65.73	1.09
HBL_3	25%	40.45	53.85	47.13	3.69
	50%	49.72	56.39	52.47	1.68
	75%	57.46	64.53	61.19	1.60
	100%	76.09	79.72	77.49	0.92

Table B.40: Cohen d Effect Size Table for Musk V2

Algorithm	Dimension	RP	KNN	ID3	HBL_3
BFG	25%	0.56	NS	0.73	0.61
	50%	0.61	NS	0.61	NS
	75%	0.45	NS	0.88	0.96
	100%	NS	NS	0.97	0.55
RP	25%	–	0.44	0.38	NS
	50%	–	0.58	0.86	0.64
	75%	–	0.51	0.84	0.96
	100%	–	NS	0.96	0.58
KNN	25%	–	–	0.66	0.49
	50%	–	–	0.73	0.29
	75%	–	–	0.90	0.97
	100%	–	–	0.97	0.63
ID3	25%	–	–	–	0.34
	50%	–	–	–	0.78
	75%	–	–	–	0.69
	100%	–	–	–	0.99

B.3 Experiment Three : General Performance Evaluation

Table B.41: Iris: Neural Network Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	4.99 ± 0.03	0.01 ± 0.03	0.00 ± 0.00
	class 2	0.01 ± 0.04	4.68 ± 0.08	0.31 ± 0.08
	class 3	0.01 ± 0.03	0.19 ± 0.10	4.80 ± 0.10

Table B.42: Iris: ID3 Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	4.96 ± 0.07	0.03 ± 0.05	0.01 ± 0.04
	class 2	0.06 ± 0.07	4.60 ± 0.13	0.34 ± 0.10
	class 3	0.04 ± 0.06	0.21 ± 0.11	4.75 ± 0.09

Table B.43: Iris: KNN Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	4.95 ± 0.07	0.03 ± 0.06	0.01 ± 0.05
	class 2	0.07 ± 0.09	4.60 ± 0.11	0.33 ± 0.10
	class 3	0.03 ± 0.05	0.21 ± 0.10	4.76 ± 0.09

Table B.44: Iris: HBL_1 Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	4.87 ± 0.11	0.09 ± 0.11	0.03 ± 0.05
	class 2	0.13 ± 0.11	4.47 ± 0.13	0.41 ± 0.11
	class 3	0.08 ± 0.08	0.28 ± 0.15	4.64 ± 0.16

Table B.45: Iris: HBL_3 Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	4.97 ± 0.05	0.01 ± 0.03	0.02 ± 0.04
	class 2	0.02 ± 0.05	4.66 ± 0.09	0.32 ± 0.09
	class 3	0.01 ± 0.03	0.19 ± 0.10	4.79 ± 0.10

Table B.46: Iris: Kappa Test Results

Algorithm	Cohen's Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.95	0.01	0.94-0.96	51.92	1
KNN	0.93	0.01	0.92-0.95	51.01	1
ID3	0.93	0.01	0.92-0.95	50.97	1
HBL_1	0.90	0.01	0.88-0.92	49.25	1
HBL_3	0.94	0.01	0.93-0.96	51.63	1

Table B.47: Glass:ID3 Confusion matrix

		Predicted Classes					
		class 1	class 2	class 3	class 4	class 5	class 6
	class 1	5.99 ± 0.25	0.87 ± 0.21	0.04 ± 0.09	0.04 ± 0.07	0.04 ± 0.08	0.01 ± 0.03
	class 2	2.95 ± 0.32	4.45 ± 0.35	0.04 ± 0.07	0.07 ± 0.08	0.05 ± 0.07	0.04 ± 0.06
	class 3	1.20 ± 0.22	0.16 ± 0.14	0.26 ± 0.21	0.02 ± 0.05	0.04 ± 0.06	0.02 ± 0.06
	class 4	0.16 ± 0.16	0.16 ± 0.13	0.06 ± 0.09	0.79 ± 0.21	0.06 ± 0.10	0.07 ± 0.09
	class 5	0.37 ± 0.21	0.11 ± 0.11	0.03 ± 0.06	0.04 ± 0.11	0.31 ± 0.21	0.03 ± 0.07
	class 6	0.29 ± 0.12	0.15 ± 0.12	0.06 ± 0.10	0.05 ± 0.07	0.04 ± 0.06	2.32 ± 0.24

Table B.48: Glass: Neural Network Confusion matrix

		Predicted Classes					
		class 1	class 2	class 3	class 4	class 5	class 6
	class 1	6.01 ± 0.19	0.97 ± 0.19	0.02 ± 0.04	0.00 ± 0.00	0.00 ± 0.00	0.01 ± 0.03
	class 2	3.08 ± 0.29	4.27 ± 0.31	0.04 ± 0.06	0.11 ± 0.09	0.07 ± 0.08	0.02 ± 0.04
	class 3	1.36 ± 0.10	0.33 ± 0.10	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.02	0.00 ± 0.00
	class 4	0.24 ± 0.12	0.32 ± 0.09	0.03 ± 0.05	0.64 ± 0.15	0.01 ± 0.03	0.07 ± 0.07
	class 5	0.48 ± 0.19	0.16 ± 0.12	0.03 ± 0.05	0.04 ± 0.05	0.17 ± 0.10	0.02 ± 0.04
	class 6	0.32 ± 0.10	0.20 ± 0.11	0.05 ± 0.07	0.07 ± 0.08	0.06 ± 0.07	2.21 ± 0.12

Table B.49: Glass: KNN Confusion matrix

		Predicted Classes					
		class 1	class 2	class 3	class 4	class 5	class 6
	class 1	6.23 ± 0.23	0.68 ± 0.24	0.04 ± 0.06	0.03 ± 0.06	0.01 ± 0.03	0.02 ± 0.04
	class 2	2.73 ± 0.38	4.70 ± 0.35	0.03 ± 0.06	0.07 ± 0.09	0.06 ± 0.09	0.02 ± 0.04
	class 3	0.99 ± 0.25	0.10 ± 0.10	0.51 ± 0.25	0.01 ± 0.05	0.03 ± 0.08	0.05 ± 0.07
	class 4	0.09 ± 0.10	0.09 ± 0.10	0.04 ± 0.07	1.01 ± 0.17	0.04 ± 0.06	0.03 ± 0.05
	class 5	0.18 ± 0.19	0.06 ± 0.08	0.03 ± 0.07	0.05 ± 0.09	0.55 ± 0.22	0.03 ± 0.05
	class 6	0.19 ± 0.14	0.05 ± 0.09	0.04 ± 0.09	0.02 ± 0.04	0.04 ± 0.07	2.56 ± 0.21

Table B.50: Glass: HBL_1 Confusion matrix

		Predicted Classes					
		class 1	class 2	class 3	class 4	class 5	class 6
	class 1	6.32 ± 0.29	0.59 ± 0.29	0.03 ± 0.07	0.02 ± 0.04	0.01 ± 0.03	0.02 ± 0.05
	class 2	2.65 ± 0.34	4.72 ± 0.25	0.06 ± 0.09	0.06 ± 0.11	0.04 ± 0.08	0.06 ± 0.10
	class 3	1.00 ± 0.21	0.09 ± 0.09	0.49 ± 0.22	0.03 ± 0.06	0.06 ± 0.08	0.04 ± 0.07
	class 4	0.07 ± 0.09	0.06 ± 0.06	0.03 ± 0.04	1.05 ± 0.18	0.05 ± 0.08	0.05 ± 0.07
	class 5	0.21 ± 0.18	0.08 ± 0.10	0.02 ± 0.04	0.04 ± 0.07	0.53 ± 0.17	0.02 ± 0.05
	class 6	0.14 ± 0.11	0.07 ± 0.09	0.02 ± 0.05	0.03 ± 0.06	0.02 ± 0.05	2.61 ± 0.17

Table B.51: HBL_3 Confusion matrix

		Predicted Classes					
		class 1	class 2	class 3	class 4	class 5	class 6
	class 1	6.40 ± 0.33	0.46 ± 0.30	0.04 ± 0.09	0.04 ± 0.09	0.03 ± 0.05	0.02 ± 0.04
	class 2	2.48 ± 0.40	5.02 ± 0.37	0.02 ± 0.05	0.04 ± 0.07	0.03 ± 0.05	0.01 ± 0.03
	class 3	0.81 ± 0.21	0.08 ± 0.08	0.71 ± 0.24	0.04 ± 0.10	0.03 ± 0.05	0.03 ± 0.04
	class 4	0.06 ± 0.08	0.08 ± 0.09	0.05 ± 0.09	1.05 ± 0.19	0.03 ± 0.07	0.03 ± 0.06
	class 5	0.12 ± 0.12	0.06 ± 0.11	0.02 ± 0.04	0.04 ± 0.07	0.65 ± 0.20	0.02 ± 0.04
	class 6	0.15 ± 0.15	0.03 ± 0.06	0.02 ± 0.05	0.03 ± 0.04	0.01 ± 0.03	2.66 ± 0.15

Table B.52: Glass Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.58	0.1598	0.27-0.89	3.33	1
KNN	0.69	0.1225	0.45-0.93	5.29	1
ID3	0.58	0.1598	0.27-0.89	3.33	1
HBL_1	0.66	0.1314	0.41-0.92	4.68	1
HBL_3	0.80	0.1072	0.59-1.00	5.97	1

Table B.53: Zoo: ID3 Confusion matrix

	Predicted Classes						
	class 1	class 2	class 3	class 4	class 5	class 6	class 7
class 1	3.97 ± 0.13	0.03 ± 0.06	0.01 ± 0.03	0.02 ± 0.05	0.02 ± 0.05	0.03 ± 0.05	0.02 ± 0.05
class 2	0.01 ± 0.03	1.86 ± 0.11	0.04 ± 0.06	0.01 ± 0.03	0.03 ± 0.06	0.02 ± 0.04	0.03 ± 0.04
class 3	0.04 ± 0.07	0.04 ± 0.08	0.29 ± 0.12	0.02 ± 0.04	0.04 ± 0.06	0.03 ± 0.05	0.04 ± 0.06
class 4	0.02 ± 0.04	0.03 ± 0.07	0.04 ± 0.08	1.15 ± 0.11	0.03 ± 0.05	0.02 ± 0.05	0.01 ± 0.03
class 5	0.03 ± 0.05	0.03 ± 0.04	0.02 ± 0.04	0.04 ± 0.07	0.24 ± 0.11	0.02 ± 0.04	0.02 ± 0.06
class 6	0.04 ± 0.07	0.02 ± 0.05	0.02 ± 0.04	0.03 ± 0.07	0.03 ± 0.05	0.62 ± 0.14	0.03 ± 0.05
class 7	0.05 ± 0.07	0.02 ± 0.08	0.03 ± 0.06	0.03 ± 0.07	0.01 ± 0.03	0.05 ± 0.08	0.81 ± 0.15

Table B.54: Zoo: Neural Network Confusion matrix

	Predicted Classes						
	class 1	class 2	class 3	class 4	class 5	class 6	class 7
class 1	4.01 ± 0.10	0.01 ± 0.03	0.02 ± 0.05	0.02 ± 0.05	0.01 ± 0.03	0.01 ± 0.03	0.02 ± 0.06
class 2	0.03 ± 0.05	1.96 ± 0.06	0.00 ± 0.00	0.00 ± 0.02	0.00 ± 0.02	0.00 ± 0.00	0.00 ± 0.00
class 3	0.13 ± 0.11	0.05 ± 0.07	0.19 ± 0.10	0.02 ± 0.04	0.08 ± 0.08	0.02 ± 0.04	0.02 ± 0.04
class 4	0.03 ± 0.05	0.01 ± 0.03	0.03 ± 0.05	1.21 ± 0.09	0.01 ± 0.03	0.00 ± 0.00	0.01 ± 0.03
class 5	0.03 ± 0.06	0.00 ± 0.02	0.07 ± 0.06	0.00 ± 0.02	0.29 ± 0.05	0.00 ± 0.02	0.01 ± 0.03
class 6	0.04 ± 0.05	0.00 ± 0.02	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.02	0.74 ± 0.06	0.01 ± 0.03
class 7	0.08 ± 0.08	0.02 ± 0.04	0.03 ± 0.05	0.03 ± 0.04	0.00 ± 0.02	0.09 ± 0.07	0.75 ± 0.10

Table B.55: Zoo: KNN Confusion matrix

	Predicted Classes						
	class 1	class 2	class 3	class 4	class 5	class 6	class 7
class 1	4.04 ± 0.09	0.02 ± 0.04	0.01 ± 0.03	0.02 ± 0.05	0.01 ± 0.04	0.00 ± 0.00	0.01 ± 0.03
class 2	0.01 ± 0.03	1.94 ± 0.08	0.00 ± 0.02	0.02 ± 0.05	0.01 ± 0.03	0.02 ± 0.04	0.00 ± 0.02
class 3	0.01 ± 0.03	0.00 ± 0.02	0.42 ± 0.08	0.02 ± 0.04	0.02 ± 0.05	0.02 ± 0.04	0.01 ± 0.03
class 4	0.01 ± 0.04	0.01 ± 0.03	0.01 ± 0.03	1.25 ± 0.06	0.00 ± 0.00	0.00 ± 0.02	0.02 ± 0.05
class 5	0.01 ± 0.03	0.01 ± 0.03	0.01 ± 0.03	0.01 ± 0.04	0.35 ± 0.08	0.00 ± 0.00	0.01 ± 0.03
class 6	0.01 ± 0.03	0.00 ± 0.00	0.01 ± 0.03	0.01 ± 0.03	0.01 ± 0.03	0.75 ± 0.06	0.01 ± 0.03
class 7	0.02 ± 0.04	0.01 ± 0.03	0.02 ± 0.04	0.01 ± 0.03	0.01 ± 0.04	0.02 ± 0.05	0.92 ± 0.09

Table B.56: Zoo: HBL_1 Confusion matrix

	Predicted Classes						
	class 1	class 2	class 3	class 4	class 5	class 6	class 7
class 1	4.01 ± 0.08	0.02 ± 0.06	0.00 ± 0.02	0.02 ± 0.05	0.01 ± 0.03	0.02 ± 0.05	0.01 ± 0.03
class 2	0.01 ± 0.03	1.90 ± 0.12	0.03 ± 0.05	0.00 ± 0.00	0.01 ± 0.03	0.04 ± 0.07	0.01 ± 0.03
class 3	0.03 ± 0.06	0.01 ± 0.03	0.39 ± 0.09	0.01 ± 0.03	0.02 ± 0.04	0.01 ± 0.03	0.03 ± 0.05
class 4	0.01 ± 0.03	0.01 ± 0.03	0.03 ± 0.06	1.19 ± 0.09	0.03 ± 0.05	0.01 ± 0.03	0.03 ± 0.05
class 5	0.02 ± 0.04	0.04 ± 0.09	0.02 ± 0.05	0.02 ± 0.04	0.29 ± 0.11	0.01 ± 0.06	0.00 ± 0.02
class 6	0.01 ± 0.03	0.01 ± 0.03	0.01 ± 0.03	0.01 ± 0.03	0.03 ± 0.06	0.72 ± 0.09	0.01 ± 0.03
class 7	0.03 ± 0.06	0.00 ± 0.02	0.01 ± 0.03	0.01 ± 0.04	0.01 ± 0.03	0.03 ± 0.06	0.91 ± 0.11

Table B.57: Zoo: HBL_3 Confusion matrix

	Predicted Classes						
	class 1	class 2	class 3	class 4	class 5	class 6	class 7
class 1	4.04 ± 0.09	0.00 ± 0.02	0.01 ± 0.06	0.01 ± 0.03	0.02 ± 0.05	0.00 ± 0.00	0.02 ± 0.04
class 2	0.01 ± 0.03	1.93 ± 0.08	0.01 ± 0.03	0.02 ± 0.05	0.01 ± 0.03	0.01 ± 0.03	0.01 ± 0.03
class 3	0.03 ± 0.05	0.01 ± 0.04	0.45 ± 0.07	0.00 ± 0.02	0.01 ± 0.03	0.01 ± 0.03	0.00 ± 0.00
class 4	0.00 ± 0.02	0.01 ± 0.04	0.00 ± 0.02	1.26 ± 0.07	0.02 ± 0.04	0.01 ± 0.03	0.00 ± 0.02
class 5	0.01 ± 0.03	0.00 ± 0.00	0.01 ± 0.03	0.00 ± 0.02	0.37 ± 0.06	0.00 ± 0.02	0.01 ± 0.03
class 6	0.00 ± 0.02	0.01 ± 0.03	0.02 ± 0.05	0.01 ± 0.03	0.00 ± 0.02	0.75 ± 0.07	0.00 ± 0.02
class 7	0.02 ± 0.05	0.01 ± 0.03	0.01 ± 0.03	0.01 ± 0.03	0.00 ± 0.02	0.01 ± 0.03	0.94 ± 0.06

Table B.58: Zoo Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.87	0.1225	0.63-1.00	5.51	1
KNN	0.94	0.0840	0.78-1.00	6.05	1
ID3	0.85	0.1317	0.59-1.00	5.43	1
HBL_1	0.91	0.1046	0.71-1.00	5.83	1
HBL_3	0.95	0.0772	0.80-1.00	6.11	1

Table B.59: Wine: Neural Network Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	5.67 ± 0.07	0.23 ± 0.07	0.00 ± 0.00
	class 2	0.79 ± 0.23	6.09 ± 0.24	0.22 ± 0.09
	class 3	0.12 ± 0.09	0.23 ± 0.13	4.46 ± 0.14

Table B.60: Wine: KNN Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	5.35 ± 0.34	0.34 ± 0.26	0.21 ± 0.22
	class 2	0.75 ± 0.35	5.89 ± 0.49	0.46 ± 0.36
	class 3	0.25 ± 0.18	0.39 ± 0.27	4.16 ± 0.34

Table B.61: Wine: ID3 Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	4.01 ± 0.46	1.29 ± 0.34	0.60 ± 0.29
	class 2	1.75 ± 0.41	4.10 ± 0.48	1.24 ± 0.34
	class 3	0.58 ± 0.25	1.27 ± 0.44	2.95 ± 0.42

Table B.62: Wine: HBL_1 Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	5.40 ± 0.24	0.32 ± 0.24	0.18 ± 0.18
	class 2	0.82 ± 0.38	6.01 ± 0.34	0.27 ± 0.20
	class 3	0.17 ± 0.19	0.28 ± 0.21	4.34 ± 0.25

Table B.63: Wine: HBL_3 Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	5.45 ± 0.28	0.31 ± 0.26	0.14 ± 0.17
	class 2	0.54 ± 0.27	6.27 ± 0.37	0.30 ± 0.28
	class 3	0.16 ± 0.15	0.25 ± 0.25	4.39 ± 0.28

Table B.64: Wine Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.87	0.1022	0.66-1.00	5.10	1
KNN	0.80	0.1224	0.56-1.00	4.71	1
ID3	0.43	0.1742	0.09-0.77	2.52	1
HBL_1	0.83	0.1146	0.60-1.00	4.88	1
HBL_3	0.86	0.1055	0.65-1.00	5.07	1

Table B.65: Parkinson: ID3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	14.96 ± 0.38	1.54 ± 0.38
	class 2	2.26 ± 0.32	0.74 ± 0.32

Table B.66: Parkinson: KNN Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	15.15 ± 0.37	1.35 ± 0.37
	class 2	2.16 ± 0.44	0.84 ± 0.44

Table B.67: Parkinson: HBL_1 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	14.49 ± 0.48	2.01 ± 0.48
	class 2	2.39 ± 0.42	0.61 ± 0.42

Table B.68: Parkinson: HBL_3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	15.53 ± 0.36	0.97 ± 0.36
	class 2	2.11 ± 0.33	0.89 ± 0.33

Table B.69: Parkinson: Neural Network Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	15.93 ± 0.26	0.57 ± 0.26
	class 2	1.67 ± 0.24	1.33 ± 0.24

Table B.70: Parkinson Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.49	0.3259	0.0-1.00	1.96	1
KNN	0.22	0.3756	0.0-0.96	0.94	0
ID3	0.17	0.3821	0.0-0.92	0.73	0
HBL_1	0.08	0.3837	0.0-0.84	0.37	0
HBL_3	0.28	0.3760	0.0-1.00	1.00	0

Table B.71: Pima: ID3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	42.76 ± 0.73	7.24 ± 0.73
	class 2	14.50 ± 0.67	12.30 ± 0.67

Table B.72: Pima: KNN Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	43.08 ± 0.74	6.92 ± 0.74
	class 2	14.58 ± 0.73	12.22 ± 0.73

Table B.73: Pima: HBL_1 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	41.15 ± 0.57	8.85 ± 0.57
	class 2	16.57 ± 0.71	10.23 ± 0.71

Table B.74: Pima: HBL_3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	44.04 ± 0.76	5.96 ± 0.76
	class 2	13.64 ± 0.68	13.16 ± 0.68

Table B.75: Pima: Neural Network Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	44.39 ± 0.53	5.61 ± 0.53
	class 2	13.20 ± 0.45	13.60 ± 0.45

Table B.76: Pima Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.42	0.1156	0.20-0.65	3.44	1
KNN	0.34	0.1208	0.10-0.58	2.76	1
ID3	0.34	0.1207	0.10-0.57	2.74	1
HBL_1	0.22	0.1266	0.00-0.47	1.78	0
HBL_3	0.40	0.1173	0.17-0.63	3.23	1

Table B.77: Breast Cancer: KNN Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	43.60 ± 0.64	2.20 ± 0.64
	class 2	2.16 ± 0.44	21.94 ± 0.44

Table B.78: Breast Cancer: HBL_1 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	42.90 ± 0.57	2.90 ± 0.57
	class 2	2.53 ± 0.67	21.57 ± 0.67

Table B.79: Breast Cancer: HBL_3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	43.47 ± 0.54	2.33 ± 0.54
	class 2	1.98 ± 0.44	22.12 ± 0.44

Table B.80: Breast Cancer: Neural Network Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	44.34 ± 0.14	1.46 ± 0.14
	class 2	1.13 ± 0.21	22.97 ± 0.21

Table B.81: Breast Cancer Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.92	0.0498	0.82-1.00	7.68	1
KNN	0.86	0.0640	0.74-0.99	7.21	1
HBL_1	0.83	0.0706	0.69-0.97	6.93	1
HBL_3	0.86	0.0635	0.74-0.99	7.22	1

Table B.82: Madelon: ID3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	82.00 ± 4.64	48.00 ± 4.64
	class 2	76.41 ± 5.15	53.59 ± 5.15

Table B.83: Madelon: KNN Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	106.18 ± 4.99	23.82 ± 4.99
	class 2	52.06 ± 5.14	77.94 ± 5.14

Table B.84: Madelon: HBL_1 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	102.85 ± 5.27	27.15 ± 5.27
	class 2	55.58 ± 5.51	74.42 ± 5.51

Table B.95: Wine: Neural Network Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	5.67 ± 0.07	0.23 ± 0.07	0.00 ± 0.00
	class 2	0.79 ± 0.23	6.09 ± 0.24	0.22 ± 0.09
	class 3	0.12 ± 0.09	0.23 ± 0.13	4.46 ± 0.14

Table B.97: Wine: ID3 Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	4.01 ± 0.46	1.29 ± 0.34	0.60 ± 0.29
	class 2	1.75 ± 0.41	4.10 ± 0.48	1.24 ± 0.34
	class 3	0.58 ± 0.25	1.27 ± 0.44	2.95 ± 0.42

Table B.85: Madelon: HBL_3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	107.87 ± 4.42	22.13 ± 4.42
	class 2	50.16 ± 5.25	79.84 ± 5.25

Table B.86: Madelon: Neural Network Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	87.03 ± 4.48	42.97 ± 4.48
	class 2	71.38 ± 5.05	58.62 ± 5.05

Table B.87: Madelon Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.12	0.0616	0.00-0.24	1.82	0
KNN	0.42	0.0564	0.31-0.53	6.28	1
HBL_1	0.36	0.0578	0.25-0.48	5.48	1
HBL_3	0.44	0.0556	0.33-0.55	6.71	1

Table B.98: Wine: HBL_1 Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	5.40 ± 0.24	0.32 ± 0.24	0.18 ± 0.18
	class 2	0.82 ± 0.38	6.01 ± 0.34	0.27 ± 0.20
	class 3	0.17 ± 0.19	0.28 ± 0.21	4.34 ± 0.25

Table B.99: Wine: HBL_3 Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	5.45 ± 0.28	0.31 ± 0.26	0.14 ± 0.17
	class 2	0.54 ± 0.27	6.27 ± 0.37	0.30 ± 0.28
	class 3	0.16 ± 0.15	0.25 ± 0.25	4.39 ± 0.28

Table B.88: Musk V2: ID3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	13.1 ± 0.83	88.6 ± 0.83
	class 2	137.2 ± 0.55	420.9 ± 0.55

Table B.89: Musk V2: KNN Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	16.1 ± 0.92	85.6 ± 0.92
	class 2	83.6 ± 0.49	474.5 ± 0.49

Table B.90: Musk V2: HBL_1 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	13.5 ± 0.91	88.2 ± 0.91
	class 2	99.7 ± 0.71	458.4 ± 0.71

Table B.100: Wine Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.87	0.1022	0.66-1.00	5.10	1
KNN	0.80	0.1224	0.56-1.00	4.71	1
ID3	0.43	0.1742	0.09-0.77	2.52	1
HBL_1	0.83	0.1146	0.60-1.00	4.88	1
HBL_3	0.86	0.1055	0.65-1.00	5.07	1

Table B.101: Parkinson: ID3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	14.96 ± 0.38	1.54 ± 0.38
	class 2	2.26 ± 0.32	0.74 ± 0.32

Table B.91: Madelon: HBL_3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	29.6 ± 4.42	72.1 ± 4.42
	class 2	76.1 ± 5.25	482 ± 5.25

Table B.92: Neural Network Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	99.26 ± 0.72	2.44 ± 0.72
	class 2	2.31 ± 0.51	555.79 ± 0.51

Table B.93: Musk V2 Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.13	0.0656	0.00-0.10	0.81	0
KNN	0.01	0.0657	0.00-0.14	0.23	0
ID3	0.10	0.0593	0.00-0.20	2.33	1
HBL_1	0.04	0.0644	0.17-0.08	1.12	0
HBL_3	0.15	0.0613	0.03-0.27	3.91	1

Table B.102: Parkinson: KNN Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	15.15 ± 0.37	1.35 ± 0.37
	class 2	2.16 ± 0.44	0.84 ± 0.44

Table B.103: Parkinson: HBL_1 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	14.49 ± 0.48	2.01 ± 0.48
	class 2	2.39 ± 0.42	0.61 ± 0.42

Table B.94: Zoo Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.87	0.1225	0.63-1.00	5.51	1
KNN	0.94	0.0840	0.78-1.00	6.05	1
ID3	0.85	0.1317	0.59-1.00	5.43	1
HBL_1	0.91	0.1046	0.71-1.00	5.83	1
HBL_3	0.95	0.0772	0.80-1.00	6.11	1

Table B.96: Wine: KNN Confusion matrix

		Predicted Classes		
		class 1	class 2	class 3
	class 1	5.35 ± 0.34	0.34 ± 0.26	0.21 ± 0.22
	class 2	0.75 ± 0.35	5.89 ± 0.49	0.46 ± 0.36
	class 3	0.25 ± 0.18	0.39 ± 0.27	4.16 ± 0.34

Table B.104: Parkinson: HBL_3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	15.53 ± 0.36	0.97 ± 0.36
	class 2	2.11 ± 0.33	0.89 ± 0.33

Table B.105: Parkinson: Neural Network Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	15.93 ± 0.26	0.57 ± 0.26
	class 2	1.67 ± 0.24	1.33 ± 0.24

Table B.106: Parkinson Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.48	0.3259	-0.16-1.12	1.95	0
KNN	0.22	0.3756	-0.51-0.96	0.94	0
ID3	0.17	0.3821	-0.58-0.92	0.73	0
HBL_1	0.08	0.3837	-0.67-0.84	0.37	0
HBL_3	0.28	0.3760	-0.46-1.02	1.00	0

Table B.107: Pima: ID3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	42.76 ± 0.73	7.24 ± 0.73
	class 2	14.50 ± 0.67	12.30 ± 0.67

Table B.108: Pima: KNN Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	43.08 ± 0.74	6.92 ± 0.74
	class 2	14.58 ± 0.73	12.22 ± 0.73

Table B.109: Pima: HBL_1 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	41.15 ± 0.57	8.85 ± 0.57
	class 2	16.57 ± 0.71	10.23 ± 0.71

Table B.110: Pima: HBL_3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	44.04 ± 0.76	5.96 ± 0.76
	class 2	13.64 ± 0.68	13.16 ± 0.68

Table B.111: Pima: Neural Network Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	44.39 ± 0.53	5.61 ± 0.53
	class 2	13.20 ± 0.45	13.60 ± 0.45

Table B.112: Pima Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.42	0.1156	0.20-0.65	3.44	1
KNN	0.34	0.1208	0.10-0.58	2.76	1
ID3	0.34	0.1207	0.10-0.57	2.74	1
HBL_1	0.22	0.1266	-0.03-0.47	1.78	0
HBL_3	0.40	0.1173	0.17-0.63	3.23	1

Table B.113: Breast Cancer: KNN Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	43.60 ± 0.64	2.20 ± 0.64
	class 2	2.16 ± 0.44	21.94 ± 0.44

Table B.114: Breast Cancer: HBL_1 Confusion matrix

	Predicted Classes		
		class 1	class 2
	class 1	42.90 ± 0.57	2.90 ± 0.57
	class 2	2.53 ± 0.67	21.57 ± 0.67

Table B.115: Breast Cancer: HBL_3 Confusion matrix

	Predicted Classes		
		class 1	class 2
	class 1	43.47 ± 0.54	2.33 ± 0.54
	class 2	1.98 ± 0.44	22.12 ± 0.44

Table B.116: Breast Cancer: Neural Network Confusion matrix

	Predicted Classes		
		class 1	class 2
	class 1	44.34 ± 0.14	1.46 ± 0.14
	class 2	1.13 ± 0.21	22.97 ± 0.21

Table B.117: Breast Cancer Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.92	0.0498	0.82-1.02	7.68	1
KNN	0.86	0.0640	0.74-0.99	7.21	1
HBL_1	0.83	0.0706	0.69-0.97	6.93	1
HBL_3	0.86	0.0635	0.74-0.99	7.22	1

Table B.118: Madelon: ID3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	82.00 ± 4.64	48.00 ± 4.64
	class 2	76.41 ± 5.15	53.59 ± 5.15

Table B.119: Madelon: KNN Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	106.18 ± 4.99	23.82 ± 4.99
	class 2	52.06 ± 5.14	77.94 ± 5.14

Table B.120: Madelon: HBL_1 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	102.85 ± 5.27	27.15 ± 5.27
	class 2	55.58 ± 5.51	74.42 ± 5.51

Table B.121: Madelon: HBL_3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	107.87 ± 4.42	22.13 ± 4.42
	class 2	50.16 ± 5.25	79.84 ± 5.25

Table B.122: Madelon: Neural Network Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	87.03 ± 4.48	42.97 ± 4.48
	class 2	71.38 ± 5.05	58.62 ± 5.05

Table B.123: Madelon Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.12	0.0616	-0.00-0.24	1.82	0
KNN	0.42	0.0564	0.31-0.53	6.28	1
HBL_1	0.36	0.0578	0.25-0.48	5.48	1
HBL_3	0.44	0.0556	0.33-0.55	6.71	1

Table B.124: Musk V2: ID3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	94.37 ± 0.83	7.33 ± 0.83
	class 2	7.52 ± 0.55	550.58 ± 0.55

Table B.125: Musk V2: KNN Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	94.31 ± 0.92	7.39 ± 0.92
	class 2	7.46 ± 0.49	550.64 ± 0.49

Table B.126: Musk V2: HBL_1 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	94.10 ± 0.91	7.60 ± 0.91
	class 2	7.25 ± 0.71	550.85 ± 0.71

Table B.127: Musk V2: HBL_3 Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	94.27 ± 0.88	7.43 ± 0.88
	class 2	7.42 ± 0.70	550.68 ± 0.70

Table B.128: Neural Network Confusion matrix

		Predicted Classes	
		class 1	class 2
	class 1	99.26 ± 0.72	2.44 ± 0.72
	class 2	2.31 ± 0.51	555.79 ± 0.51

Table B.129: Musk V2 Kappa Test Results

Algorithm	Cohens Kappa	Kappa Error	Confidence Interval	Z Score	Stat Result
RP	0.97	0.0126	0.95-1.00	24.98	1
KNN	0.91	0.0221	0.87-0.96	23.47	1
HBL_1	0.91	0.0222	0.87-0.96	23.47	1
HBL_3	0.91	0.0221	0.87-0.96	23.47	1