Efficient and Accurate kNN based Classification and Regression

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

MASTERS OF SCIENCE BY RESEARCH
in
COMPUTER SCIENCE

by

Harshit Dubey
200802014
harshit.dubeyug08@students.iiit.ac.in

CENTER FOR DATA ENGINEERING
International Institute of Information Technology
Hyderabad - 500 032, INDIA
March 2013

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Efficient and Accurate kNN based Classification and Regression" by Harshit Dubey, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Dr. Vikram Pudi

To My Family and Friends

Acknowledgments

I would like to take this opportunity to acknowledge and appreciate the efforts of the people who have helped me during my research and documenting this thesis. First and foremost, I would like to express my deep-felt gratitude towards my thesis adviser, Dr. Vikram Pudi for his excellent guidance and invaluable support. His invaluable and continuous exposure, not only to research but also to other aspects of life have helped in constructive development and shaping of my ideologies and understanding.

I would like to offer my gratitude to my friends Aditya, Ajay, Ankit, Mihir, Mohit, Sanidhya, Saumay, Varun and Vipul for the fruitful discussions and constant motivation. I am very thankful to my seniours, specially Aditya Desai, Himanshu Singh and Raghvendra Mall for providing continuous guidance through out my thesis work.

I take this opportunity to thank my parents and grand parents for their endless love, support and motivation. They have been with me in my ups and downs, and kept encouraging me. I also thank my brother for his support and encouragement.

Abstract

A K-Nearest Neighbor based classifier classifies a query instance based on the class labels of its neighbor instances. Although kNN has proved to be a ubiquitous classification/regression tool with good scalability, but it suffers from some drawbacks. Two of its major drawbacks are: (1) The existing kNN algorithm is equivalent to using only local prior probabilities to predict instance labels, and hence it does not take into account the class distribution around the wider neighborhood of the query instance, which results into undesirable performance on imbalanced data. (2) It uses all the training data at the runtime and hence is slow. In this thesis, we provide a suite of solutions to tackle this drawbacks.

BINER: We propose a new efficient technique for regression. Our algorithm instead of directly predicting the value of response variable, narrows down the range in which the response variable has the maximum likelihood of occurrence and then interpolates to give the output. The data is Hierarchically partitioned in the pre processing step, and search for the partition in which the response has the maximum likelihood of occurrence is carried out at the runtime. It takes a single parameter k, the same as in kNN and more than often outperforms the conventional state of art methods on a wide variety of datasets as illustrated by our experimental study.

CLUEKR: We propose a novel, efficient and outlier resistant, clustering based kNN regression algorithm CLUEKR, which instead of searching for nearest neighbors directly in the entire dataset, first find the cluster in which the query point should lie. We first Hierarchically cluster the data in the pre processing step, then a recursive search starting from root node of the hierarchy is performed. At current search node in the hierarchy, we select a cluster among its child, in which the query point has maximum likelihood of occurrence and then a recursive search is applied to it. Finally we find the k nearest neighbors of query points in that cluster and return the weighted mean of their response variable as result. We also propose the modification that enable CLUEKR to be applied for classification task.

Class Based Weighted K-Nearest Neighbor: We propose a Class based Weighted K-Nearest Neighbor algorithm in which a weight is assigned to each of the class based on how its instances are classified in the neighborhood of query instance by the regular K-Nearest Neighbor classifier. The modified algorithm takes into account class distribution around the neighborhood of query instance. We ensure that the weights assigned do not give undue advantage to outliers. A thorough experimental study of the proposed approach over several real world dataset confirms that our approach performs better than the current state-of-the-art approaches.

Finally, we integrate Class Based Weighted K-Nearest Neighbor work with CLUEKR and present an efficient and accurate kNN based classifier that takes into account the nature of data.

Contents

Chapter Page

1 Introduction . . . 1
1.1 Classification . . . 1
1.2 Regression Analysis . . . 2
1.3 Imbalance Data . . . 3
1.4 Problem Formulation . . . 4
1.4.1 Regression Analysis . . . 4
1.4.2 Classification . . . 5
1.4.3 Mathematical Model of kNN . . . 5
1.5 Contributions . . . 6
1.6 Organization of Thesis . . . 7

2 Related Work . . . 9
2.1 Regression Algorithms . . . 9
2.1.1 Traditional Statistical Approaches . . . 9
2.1.2 Generalized Projection Pursuit Regression . . . 9
2.1.3 Neural Networks and Support Vector Machines . . . 10
2.1.4 Decision Tree . . . 10
2.1.5 Indexing based approaches . . . 11
2.1.6 Segmented or piecewise regression . . . 11
2.1.7 Nearest Neighbor . . . 11
2.2 Learning on Imbalance Data . . . 12
2.2.1 Decision Tree . . . 12
2.2.2 Support Vector Machines . . . 13
2.2.3 Nearest Neighbor . . . 13
2.3 Discussion . . . 15

3 BINER : BINary search based Efficient Regression . . . 16
3.1 Intuition and Methodology of BINER . . . 16
3.2 The BINER Algorithm . . . 17
3.2.1 Complexity Analysis . . . 18
3.3 Experimental Study . . . 19
3.3.1 Performance Model . . . 19
3.3.2 Results . . . 20
3.3.3 Discussion . . . 22
3.4 Conclusions . . . 22

4 CLUEKR : CLUstering based Efficient kNN Regression . . . 23
4.1 The CLUEKR Algorithm . . . 23
4.1.1 Pre-Processing . . . 23
4.1.2 Actual Algorithm . . . 24
4.1.3 Complexity Analysis . . . 24
4.2 Experimental Study . . . 25
4.2.1 Performance Model . . . 25
4.2.2 Results and Discussion . . . 26
4.3 CLUEKR for Classification Task . . . 26
4.3.1 Algorithm . . . 26
4.3.2 Experimental Study . . . 28
4.3.2.1 Performance Model . . . 28
4.3.2.2 Results and Discussion . . . 28
4.4 Conclusion . . . 29

5 Class Based Weighted K-Nearest Neighbor Over Imbalance Dataset . . . 30
5.1 An Experimental Study dealing with the design of Weighting Factor . . . 30
5.1.1 Design 1 . . . 30
5.1.2 Design 2 . . . 31
5.1.3 Design 3 . . . 32
5.1.4 Experimental Study . . . 33
5.1.4.1 Performance Model . . . 33
5.1.4.2 Results . . . 34
5.1.4.3 Discussion . . . 34
5.1.5 Conclusion . . . 35
5.2 Our Proposed Algorithm . . . 35
5.2.1 Properties of Weighting Factor . . . 36
5.2.2 Complexity Analysis . . . 38
5.3 Experimental Study . . . 38
5.3.1 Performance Model . . . 38
5.3.2 Results and Discussion . . . 39
5.4 Conclusion . . . 41

6 Integrating Class Based Weighted K-Nearest Neighbor approach with CLUEKR . . . 42
6.1 Background . . . 42
6.2 Integration Step . . . 43
6.3 Experimental Study . . . 44
6.3.1 Performance Model . . . 44
6.3.2 Results and Discussion . . . 44
6.4 Conclusion . . . 44

7 Conclusions . . . 47
7.1 Contribution . . . 47
7.2 Future Work . . . 48

Bibliography . . . 51

List of Figures

Figure Page

1.1 A sample scenario where regular kNN algorithm will fail 4

3.1 Comparison of run times of kNN and BINER 19

5.1 A sample scenario where regular kNN algorithm will fail 31
5.2 A sample scenario where data points are present in clusters with each cluster having 1 major class. 33
5.3 A sample scenario where data points are present in clusters (dotted circle represent clusters containing minority class) with each cluster having one major class. 36
5.4 Performance Comparison between Our Algorithm and kNN 40

List of Tables

Table Page

3.1 Dataset Description 20
3.2 Comparison of results of BINER and other standard approaches 21
3.3 Comparison of results of BINER and other standard approaches 21

4.1 Dataset Description 26
4.2 Comparison of results of CLUEKR with other standard approaches 27
4.3 Comparison of dataset size with size of the cluster obtained 27
4.4 Dataset Description 28
4.5 Comparison of results of CLUEKR (for classification task) with kNN 29

5.1 Dataset Description 34
5.2 Experimental results over several real world dataset 34
5.3 Dataset Description 39
5.4 Experimental results over several real world dataset 40

6.1 Dataset Description 44
6.2 Experimental results over several real world dataset 45
6.3 Comparision of Final cluster size obtained with dataset size 45

Chapter 1

Introduction

Classification has been an age old problem. Early in the 4th century BC, Aristotle tried to group organisms into two classes depending on whether they are beneficial or harmful to a human. He also introduced the concept of classifying all forms of life for organizing the rich diversity in living organisms. Today classification systems find differentiating features between classes and use them to classify unknown instances. It has been recognized as an important problem in data mining [13] among other knowledge discovery tasks.

In the recent past, a lot of research centered at nearest neighbor methodology has been done [23, 39, 49, 29, 14, 25, 41, 26]. Although kNN is computationally expensive, it is very simple to understand, accurate, requires only a few parameters to be tuned and is robust with regard to the search space. Also kNN classifier can be updated at a very little cost as new training instances with known classes are presented. A strong point of kNN is that, for all data distributions, its probability of error is bounded above by twice the Bayes probability of error [31]. However it suffers from two major drawbacks:

1. It uses only *local* prior probabilities to predict instance labels, and hence does not take into account, class distribution around the neighborhood of query instance. This results in undesirable performance on imbalanced data sets. The performance of kNN algorithm over imbalanced datasets can be improved, if it uses this information while classifying instances.

2. It is a lazy learner i.e. it uses *all* the training data at the runtime, and is hence slow.

In this thesis, we provide a suite of solutions in order to tackle the above drawbacks. In this chapter, we describe the classification problem in Section 1.1, and regression analysis in Section 1.2. Section 1.3 talks about Imbalance data, followed by Problem formulation in Section 1.4. Finally, we discuss the contributions and organization of this thesis in Section 1.5 and Section 1.6 respectively.

1.1 Classification

Classification is defined [18] as the process of finding a set of models (or functions) that describe and distinguish data classes and concepts, with the goal being to use the model to predict the classes

of objects whose class labels are unknown. Thus, classification is a supervised learning problem where the task is to predict the value of a discrete output variable given a set of training examples and a test sample where each training example is a pair consisting of the input object and the desired class.

Generally data classification is a two step process. In the first step, a classifier is built describing a pre-determined set of data classes or concepts. This is the learning step (or training phase), where a classification algorithm builds the classifier by analyzing or learning from a training set. In the second step, the model is used for classification.

Classification has been widely used in:

1. Credit scoring: Define the cap-limit for a credit card based on users past behaviour.
2. Search engines: Categorize or classify the type of query or document.
3. Handwriting recognition: Receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touch-screens and other devices.
4. Document categorization: Assign an electronic document to one or more categories, based on its contents.
5. Speech recognition and medical image analysis and diagnosis.

1.2 Regression Analysis

The problem of regression is to estimate the value of a dependent variable based on the values of one or more independent variables, e.g., predicting price increase based on demand or money supply based on inflation rate etc. Regression analysis includes any techniques for modeling and analyzing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables. More specifically, regression analysis helps to understand how the typical value of the dependent variable changes when any one of the independent variables is varied, while the other independent variables are held fixed. Regression algorithms can be used for prediction (including forecasting of time-series data), inference, hypothesis-testing and modeling of causal relationships.

Regression algorithms have been widely used for:

1. Trend Line: A trend line represents a trend or the long-term movement in time-series data.
2. Finance: For analyzing and quantifying the systematic risk of an investment.
3. Economics: Used to predict consumption spending, fixed investment spending, inventory investment, spending on imports, the demand to hold liquid assets, labor demand, and labor supply.

In statistics, Regression is considered as collection of statistical function-fitting techniques. Statistical approaches try to learn a probability function $P(y \mid x)$ and use it to predict the value of y for a

given value of x. Users study the application domain to understand the form of this probability function. The function may have multiple parameters and coefficients in its expansion. Statistical approaches although popular, are not generic in that they require the user to make an intelligent guess, about the form of regression equation, so as to get the best fit for the data.

Regression analysis has been studied extensively in statistics, but there have been only a few studies from the data mining perspective. Majority of this study resulted into algorithms that fall under the following broad categories - Linear Regression [33], Nearest Neighbor Algorithms [11], Decision Trees [4], Support Vector Machines [10], Neural Networks [19] and Logistic Regression [20]. However most of these algorithms were originally developed for classification purpose, but have been later modified for regression.

The current existing standard algorithms [33, 10, 11, 4, 19] suffer from one or more of high computational complexity, poor results, fine tuning of parameters and extensive memory requirements. kNN provides excellent accuracy, but has linear computational complexity. Finding an optimal decision tree [4] is a NP-Complete problem [22]. Neural networks are highly dependent on the initialization of weight vectors and generally, have large training time. Also, the best fit structure of the neural network has to be intelligently guessed or determined by trial and error method.

In the recent past, a lot of research centered at nearest neighbor methodology has been done. However one of the major drawbacks of kNN is that, it is a lazy learner i.e. it uses all the training data at the runtime. The accuracy of kNN highly depends upon the distance metric used. Euclidean distance is a simple and efficient method for computing distance between two reference data points. More complex distance functions may provide better results depending on the dataset and domain. But user may refrain from using a better, generally computationally more complex, distance metric due to high run time of the algorithm. This motivated us to strive for an algorithm which has a significantly low run time and hence can incorporate expensive distance metrics with ease.

1.3 Imbalance Data

Building data mining models using unreliable or abnormal datasets presents a significant challenge to classifier construction. Regardless of the strength of a particular classification algorithm, learning from poor quality data will result in sub-optimal performance. Numerous studies dealing with classification problem have shown that the presence of errors in the training dataset lowers the predictive accuracy of a learner on test data [50, 46]. There are many different dimensions of data quality, including class noise or labeling errors [24], attribute noise [43, 50], and missing values [28]. Another commonly-encountered challenge in data mining applications is the occurrence of class imbalance.

A data set is imbalanced, if its dependent variable is categorical and the number of instances in one class is different from those in the other class. In many real world applications such as Web page search, scam sites detection, fraudulent calls detection etc, there is a highly skewed distribution of classes. Various classification techniques such as kNN [12], SVM [10], and Neural Networks[19] etc

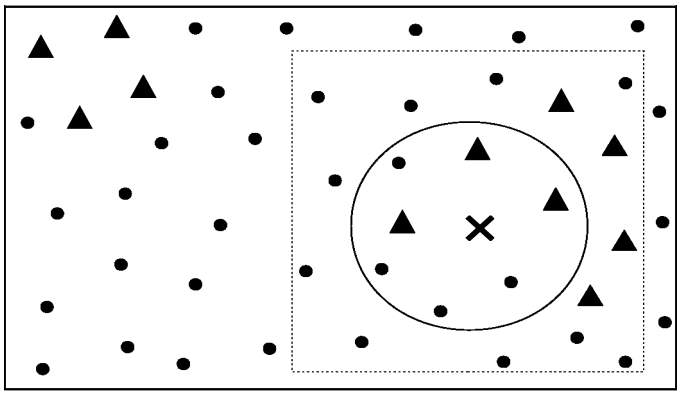

Figure 1.1 A sample scenario where regular kNN algorithm will fail

have been designed and used, but it has been observed that the algorithms do not perform as good on imbalanced datasets as on balanced datasets. Learning from imbalanced data sets has been identified as one of the 10 most challenging problems in data mining research [35]. In the literature of solving class imbalance problems, various solutions have been proposed. Such techniques broadly include two different approaches: (1) modifying existing methods or (2) application of a pre-processing stage.

In the recent past, a lot of research centered at nearest neighbor methodology has been done. However one of the major drawbacks of kNN is that, it uses only local prior probabilities to predict instance labels, and hence does not take into account, class distribution around the neighborhood of query instance. This results in undesirable performance on imbalanced data sets. The performance of kNN algorithm over imbalanced datasets can be improved, if it uses information about local class distribution while classifying instances.

Fig. 1.1 shows an artificial two-class imbalance problem, where the majority class "A" is represented by circles and the minority class "B" by triangles. The query instance is represented by cross. As can be seen from the figure, the query instance would have been classified as the majority class "A" by a regular kNN algorithm with k value equal to 7. But if the algorithm had taken into account the imbalance class distribution around the neighborhood of the query instances (say in the region represented by dotted square), it would have classified the query instance as belonging to minority class "B", which is the desired class.

1.4 Problem Formulation

1.4.1 Regression Analysis

In this subsection, we describe the problem of regression and notation used to model the dataset.

The problem of regression is to estimate the value of a dependent variable (known as response variable) based on the values of one or more independent variables (known as feature variables). We model the tuple as $\{x, y\}$ where x is an ordered set of attribute values like $\{x_1, x_2, \ldots, x_d\}$ and y is the nu-

meric variable to be predicted. Here x_i is the value of the i^{th} attribute and there are d attributes overall corresponding to a d-dimensional space.

Formally, the problem has the following inputs:

- A set of n tuples called the training dataset, $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.
- A query tuple x_t.

The output is an estimated value of y for the given query x_t. Mathematically, it can be represented as

$$y = f(x_t, D, parameters), \tag{1.1}$$

where $parameters$ are the arguments which the function $f()$ takes. These are generally set by user and are learned by trial and error method.

1.4.2 Classification

In this subsection, we describe the problem of classification and notation used to model the dataset.

The problem of classification is to estimate the value of the class variable based on the values of one or more independent variables (known as feature variables). We model the tuple as $\{x, y\}$ where x is an ordered set of attribute values like $\{x_1, x_2, \ldots, x_d\}$ and y is the class variable to be predicted. Here x_i is the value of the i^{th} attribute and there are d attributes overall corresponding to a d-dimensional space.

Formally, the problem has the following inputs:

- A set of n tuples called the training dataset, $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.
- A query tuple x_t.

The output is an estimated value of the class variable for the given query x_t, mathematically it can be expressed as:

$$y_t = f(x_t, D, parameters), \tag{1.2}$$

Where $parameters$ are the arguments that the function $f()$ takes. These are generally set by the user or are learned by some method.

1.4.3 Mathematical Model of kNN

In this subsection, we present a mathematical model for kNN algorithma and show that kNN only makes use of local prior probabilities for classification.

For a given query instance x_t, kNN algorithm works as follows:

$$y_t = \underset{c \in \{c_1, c_2, \ldots, c_m\}}{\arg\max} \sum_{x_i \in N(x_t, k)} E(y_i, c) \tag{1.3}$$

Where y_t is the predicted class for the query instance x_t and m is the number of classes present in the data. Also

$$E(a,b) = \begin{cases} 1 & if a = b \\ 0 & else \end{cases} \tag{1.4}$$

$$N(x,k) = Set\ of\ k\ nearest\ neighbor\ of\ x$$

Eq. (1.3) can also be written as

$$y_t = \arg\max \left\{ \sum_{x_i \in N(x_t,k)} E(y_i, c_1), \sum_{x_i \in N(x_t,k)} E(y_i, c_2), \quad \dots \quad , \sum_{x_i \in N(x_t,k)} E(y_i, c_m) \right\} \tag{1.5}$$

$$y_t = \arg\max \left\{ \sum_{x_i \in N(x_t,k)} \frac{E(y_i, c_1)}{k}, \sum_{x_i \in N(x_t,k)} \frac{E(y_i, c_2)}{k}, \quad \dots \quad , \sum_{x_i \in N(x_t,k)} \frac{E(y_i, c_m)}{k} \right\} \tag{1.6}$$

and we know that

$$p(c_j)_{(x_t,k)} = \sum_{x_i \in N(x_t,k)} \frac{E(y_i, c_j)}{k} \tag{1.7}$$

Where $p(c_j)_{(x_t,k)}$ is the probability of occurrence of j^{th} class in the neighborhood of x_t . Hence Eq. 1.6 turns out to be

$$y_t = \arg\max\{p(c_1)_{(x_t,k)}, p(c_2)_{(x_t,k)}, \dots, p(c_m)_{(x_t,k)}\} \tag{1.8}$$

It is clear from Eq. 1.8, that kNN algorithm uses only prior probabilities to calculate the class of the query instance. It ignores the class distribution around the neighborhood of query point.

1.5 Contributions

Following are the major contributions of this thesis:

1. BINER : We contribute a new efficient technique for regression. Our algorithm instead of directly predicting the response variable, narrows down the range in which the response variable has the maximum likelihood of occurrence and then interpolates to give the output. The data is Hierarchically partitioned in the pre-processing step, and search for the partition in which the response has the maximum likelihood of occurrence is carried out at the runtime. It takes a single parameter k, the same as in kNN and more than often outperforms the conventional state of art methods on a wide variety of datasets as illustrated in the Experimental Section.

2. CLUEKR : We propose a novel, efficient and outlier resistant, clustering based kNN regression algorithm CLUEKR, which instead of searching for nearest neighbors directly in the entire dataset, first find the cluster in which the query point should lie. We first Hierarchically cluster the data in the pre processing step, then a recursive search starting from root node of the hierarchy is performed. At current search node in the hierarchy, we select a cluster among its child, in which

the query point has maximum likelihood of occurrence and then a recursive search is applied to it. Finally we find the k nearest neighbors of query points in that cluster and return the weighted mean of their response variable as result. We also propose the modification that enable CLUEKR to be applied for classification task. One of the major advantage of CLUEKR over BINER is that, with some small modification CLUEKR can be used for classification task.

3. Class Based Weighted K-Nearest Neighbor : We propose a modified K-Nearest Neighbor algorithm to solve the imbalanced dataset classification problem. More specifically the contributions are as follows:

   (a) First we present a mathematical model of K-Nearest Neighbor algorithm and show that, it does not take into account nature of the data around the query instance.

   (b) To solve the above problem, we propose a Weighted K-Nearest Neighbor algorithm in which a weight is assigned to each of the class based on how its instances are classified in the neighborhood of query instance by the regular K-Nearest Neighbor classifier. The modified algorithm takes into account class distribution around the neighborhood of query instance. We ensure that the weights assigned do not give undue advantage to outliers.

   (c) A thorough experimental study of the proposed approach over several real world dataset was performed. The study confirms that our approach performs better than the current state-of-the-art approaches.

4. Finally, we integrate Class Based Weighted K-Nearest Neighbor work with CLUEKR and present an efficient and accurate kNN based classifier that takes into account the nature of data.

1.6 Organization of Thesis

The rest of the thesis is organized in the following manner. In Chapter 2, we look at the Related Work in the field of regression analysis and learning on imbalance dataset. We shall provide an exhaustive coverage of prominent work conducted in these areas in the past decade.

In Chapter 3 we look at the BINER algorithm and understand the intuition and methodology behind it. We evaluate its performance on diverse real world datasets and compare it with the state-of-the-art regression algorithms.

In Chapter 4 we first propose the CLUEKR algorithm for regression. We also propose the modification that enable CLUEKR to be applied for classification task. We compare CLUEKR's performance on diverse real world datasets with the state-of-the-art regression and classification algorithms.

In Chapter 5 we first perform an experimental study that deals with the design of class based weighting factor for kNN classifier. Next we propose our class based weighted K-Nearest Neighbor algorithm and look at the properties of the associated weighting factor. We evaluate our approach on various real world datasets and compare its performance with various state-of-the-art classifier.

In chapter 6 we integrate Class based Weighted kNN approach with CLUEKR (for classification) to provide an efficient and accurate kNN classifier that takes into account local class distribution during classification.

In Chapter 7 we highlight our conclusions followed by providing some ideas for the future work.

Chapter 2

Related Work

In this chapter we throw light on some related work to our thesis in the literature. In Section 2.1, we talk about the different regression algorithm in the literature. In section 2.2, we talk about the challenges involved with learning on imbalance data and the recent work done in this field. Finally in Section 2.3 we discuss the current scenario and its limitations.

2.1 Regression Algorithms

2.1.1 Traditional Statistical Approaches

Traditional Statistical approaches [45, 2] follow a "curve fitting "methodology that requires the form of the curve to be specified in advance. They try to model the relationship between the dependent and independent variables as a closed form function. It is the users responsibility to make an intelligent guess about the form of this function. This is done by studying the application domain, and may involve trial and error methods. Once the functions form is fixed, its parameters (or coefficients) are estimated so as to give a "best fit"to the available data. Typically, the function also has an error term that is used to compensate for unexplained variation in the dependent variable. This requires regression problems in each special application domain be studied and solved optimally for that domain. Another problem with these approaches is outlier (extreme cases) sensitivity.

The most common statistical regression approach is linear regression which assumes the entire data to follow a linear relationship between the response and feature variables. Surely, this naive assumption is unlikely to hold in a wide variety of applications. Variations to linear regression include Pace regression [44], Least Median Square regression among others.

2.1.2 Generalized Projection Pursuit Regression

Generalized Projection Pursuit regression [27] is a new method for statistical regression which constructs a regression surface $y = f(D)$ by estimating the form of the function f in such a manner that it best fits the dataset D. Also, the strength of the Projection Pursuit model lies in its parameterless nature

which attempts to fit a number of general smooth functions to the regression space, an improvement over existing linear regression methods where a number of linear functions are fit over the regression space. However, the given method suffers from the problem that it tries to fit the surface to the entire regression plane. Hence, it cannot capture inherent relationships for datasets where there may be both highly complex non-linearity associated in some regions while some other regions may be plateaus.

2.1.3 Neural Networks and Support Vector Machines

Neural Networks [19] is a class of data mining approaches that has been used for regression and dimensionality reduction. However, Neural Networks are complex and an in-depth analysis of results obtained is not possible.

Support Vector Machine [10] is a new data mining paradigm applied for regression. Viewing input data as two sets of vectors in an n-dimensional space, an SVM will construct a separating hyperplane in that space that maximizes the margin between the two datasets. To calculate the margin, two parallel hyperplanes are constructed, one on each side of the separating hyperplane, which are pushed up against the two data sets. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the neighboring data points of both classes, since in general the larger the margin the lower the generalization error of the classifier.

The SVM paradigm in [40, 38, 8] is referred to as SMO Regression, which implements a sequential minimal optimization algorithm for training a support vector using polynomial or RBF kernels [17]. This implementation globally replaces all missing values and transforms nominal attributes into binary ones. One of the main drawbacks of SVM is the large number of input patterns (support vectors) required for representing the optimal separating surface, resulting in algorithms of high complexity. Also, in contrast to the simplicity of the algorithms discussed in this section, SVM techniques involve complex, abstract mathematics, thus resulting in techniques that are more difficult to implement, maintain, embed and modify as the situation demands.

2.1.4 Decision Tree

Decision trees are one of the fundamental learning algorithms in the data mining community. One of the first data mining approaches to regression were regression trees [4], a variation of decision trees where the predicted output values are stored at leaf node. These nodes are finite and hence the predicted output is limited to finite set of values in contrast with the problem of predicting a continuous variable as required in regression.

The concept of decision tree is a flowchart-like tree structure, where each internal node (nonleaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf (or terminal node) holds a class label. The topmost node in a tree is the root node. The concept of decision tree has been extended so as to predict continuous (ordered) values, rather than class labels. Each

regression leaf stores a continuous-valued prediction, which is actually the average value of the predicted attribute for the training records that reach the leaf.

However decision trees primarily suffer from the problem of over-fitting the data. The pruning algorithms devised for the same require tuning of parameters through use of domain knowledge. In addition to this, these algorithms are more favorable for classification and not for predicting a continuous variable as required in regression.

2.1.5 Indexing based approaches

Roussopoulos and others presented a branch and bound R-tree traversal algorithm [37] to find nearest neighbors of a query point. The algorithm required creating and sorting an Active Branch List of Nodes [37] at each of the node and then pruning the list. Another drawback of the approach is the depth first traversal of the index that incurs unnecessary disk IO's. Berchtold et. al. [3] suggest pre-calculating, approximating and indexing the solution space for the nearest neighbor problem in d dimensional spaces. Pre-calculating the solution space means determining the Voronoi diagram of the data points. The exact Voronoi cells in d space are usually very complex, hence, the authors propose indexing approximation of the Voronoi cells. This approach is only appropriate for first nearest neighbor problem in high dimensional spaces.

2.1.6 Segmented or piecewise regression

Segmented or piecewise regression [32] is a method in regression analysis in which the independent variable is partitioned into intervals and a separate line segment is fit to each interval. It is is essentially a wedding of hierarchical clustering and standard regression theory. It can also be performed on multivariate data by partitioning the various independent variables. Segmented regression is useful when the independent variables, clustered into different groups, exhibit different relationships between the variables in these regions. The boundaries between the segments are breakpoints.

2.1.7 Nearest Neighbor

One of the oldest, accurate and simplest method for pattern classification and regression is K-Nearest-Neighbor (kNN) [11]. kNN algorithms have been identified as one of the top ten most influential data mining algorithms [47] for their ability of producing simple but powerful classifiers. It has been studied at length over the past few decades and is widely applied in many fields. Despite its simplicity, the kNN rule often yields competitive results. However one of the major drawbacks of kNN is that, it is a lazy learner i.e. it uses all the training data at the runtime.

A recent work on prototype reduction, called Weighted Distance Nearest Neighbor (WDNN) [23] is based on retaining the informative instances and learning their weights for classification. The algorithm assigns a non negative weight to each training instance tuple at the training phase and only the training

instances with positive weight are retained (as the prototypes) in the test phase. However the algorithm is specifically designed for classification purpose and cannot be used for regression.

In another recent work [39], a Parameterless, Accurate, Generic, Efficient NN-Based Regression algorithm PAGER is proposed, which is based on the assumption that value of the dependent variable varies smoothly with the variation in values of independent variable. For each dimension in the search space, the authors construct a 1-dimensional predictor as a line passing through two closest neighbor of the query point. For each of the predictor obtained, they determine the mean error occurred if the predictor was used in prediction of the k-nearest neighbors. A weight inversely proportional to the mean error is assigned to each predictor. Finally a weighted sum of the value output by individual predictors for the query instance is assigned to it.

Qi Yu and others in one of their recent work [49] proposed a methodology named Optimally Pruned K-Nearest Neighbors (OP-kNNs) which builds a one hidden-layer feed forward neural network using K-Nearest Neighbors as kernels to perform regression. The approach performed better compared to state-of-the-art methods while remaining fast.

2.2 Learning on Imbalance Data

In the literature of solving class imbalance problems, various solutions have been proposed; such techniques broadly include two different approaches, modifying methods or the application of a pre-processing stage. The pre-processing approach focuses on balancing the data, which may be done either by reducing the set of examples (undersampling) or replicate minority class examples (oversampling) [15]. One of such earliest and classic work is the SMOTE method [7] which increases the number of minor class instances by creating synthetic samples. This work is also based on the nearest neighbor analogy. The minority class is over sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors. A recent modification to SMOTE proposes that, using different weight degrees on the synthetic samples (so-called safe-level-SMOTE [6]) produces better accuracy than SMOTE. Alternative approaches that modify existing methods focus on extending or modifying the existing classification algorithms so that they can be more effective in dealing with imbalanced data. HDDT [9] and CCPDT [30] are examples of such methods, which are modification of decision tree algorithms.

2.2.1 Decision Tree

Decision trees are one of the fundamental learning algorithms in the data mining community. Decision trees, particularly C4.5 [36], have been among the more popular algorithms that have been significantly helped by sampling methods for countering the high imbalance in class distributions. The important function to consider when building a decision tree is known as the splitting criterion. This function defines how data should be split in order to maximize performance. C4.5 [36] and CART [5]

are two popular algorithms for decision tree induction; however, their corresponding splitting criteria information gain and the Gini measure are considered to be skew sensitive. It is because of this specific sensitivity to class imbalance that use of sampling methods prior to decision tree induction has become a de facto standard in the literature. The sampling methods alter the original class distribution, driving the bias towards the minority or positive class. Cieslak and Chawla in one of their work [9], outlined the strengths of Hellinger distance [34] in solving class imbalance problem and proposed its application in forming decision trees. They proved that Hellinger distance as a decision tree splitting criterion, is skew-insensitive. Finally they propose a decision tree based algorithm called HDDT, incorporating Hellinger distance as the tree splitting criterion. Their experimental studies shows that HDDT is superior compared to other decision trees under class imbalance.

In another recent work [25], Liu and others proposed a new decision tree algorithm, Class Confidence Proportion Decision Tree (CCPDT), which is robust and insensitive to size of classes and generates rules which are statistically significant. They first show that Information Gain as a tree splitting criteria, results in rules which are biased towards the majority class and then proposes a new measure, Class Confidence Proportion (CCP), which forms the basis of CCPDT.

2.2.2 Support Vector Machines

Support Vector Machine [10] is a powerful data mining paradigm applied for classification. In 2002, Tang and others [42], designed some modification to SVM, so as to appropriately tackle the problem of class imbalance. They incorporate different "rebalance" heuristics in SVM modeling including cost-sensitive learning, oversampling, and undersampling. These SVM based strategies showed improvement over the existing SVM approach. In another recent work [21], a modification to SVM was presented, in which a new weighted approach on Lagrangian support vector machine for imbalanced data classification problem is proposed. The weight parameters are embedded in the Lagrangian SVM formulation.

2.2.3 Nearest Neighbor

One of the oldest, accurate and simplest method for pattern classification and regression is K-Nearest-Neighbor (kNN) [11]. kNN algorithms have been identified as one of the top ten most influential data mining algorithms [47] for their ability of producing simple but powerful classifiers. It has been studied at length over the past few decades and is widely applied in many fields. The kNN rule classifies each unlabeled example by the majority label of its k-nearest neighbors in the training dataset. Despite its simplicity, the kNN rule often yields competitive results. A recent work on prototype reduction, called Weighted Distance Nearest Neighbor (WDNN) [23] is based on retaining the informative instances and learning their weights for classification. The algorithm assigns a non negative weight to each training instance tuple at the training phase. Only the training instances with positive weight are retained (as the prototypes) in the test phase. Although the WDNN algorithm is well formulated and

shows encouraging performance, in practice it can only work with K = 1. A more recent approach WDkNN [48] tries to reduce the time complexity of WDNN and extend it to work for values of K greater than 1.

Chawla and Liu in one of their recent work [29] presented a novel K-Nearest Neighbors weighting strategy for handling the problem of class imbalance. They proposed CCW (class confidence weights) that uses the probability of attribute values given class labels to weight prototypes in kNN. While the regular kNN directly uses the probabilities of class labels in the neighborhood of the query instance, they used conditional probabilities of classes. They have also shown how to calculate CCW weights using mixture modeling and Bayesian networks. The method performed more accurately than the existing state-of-art algorithms.

KaiYan Feng and others [14] defined a new neighborhood relationship known as passive nearest neighbors. For two points A and B belonging to class L, point B is the local passive k^{th}-order nearest neighbor of A, only and only if A is the k^{th} nearest neighbor of B among all data of class L. For each query point, its k actual nearest neighbor and k passive nearest neighbors are first calculated and based on it, a overall score is calculated for each class. The class score determines the likelihood that the query points belong to that class.

In another recent work [25], Evan and others proposes to use geometric structure of data to mitigate the effects of class imbalance. The method even works, when the level of imbalance changes in the training data, such as online streaming data. For each query point, a k dimensional vector is calculated for each of the classes present in the data. The vector consist of distances of the query point to it's k nearest neighbors in that class. Based on this vector probability that the query point belongs to a particular class is calculated. However the approach is not studied in depth.

Yang Song and others proposes [41] two different versions of kNN based on the idea of informativeness. According to them, a point is treated to be informative, if it is close to the query point and far away from the points with different class labels. One of the proposed versions LI-KNN takes two parameters k and I, It first find the k nearest neighbor of the query point and then among them it find the I most informative points. Based on the class label of the informative points, class label is assigned to the query point. They also showed that the value of k and I have very less effect on the final result. The other version GI-KNN works on the assumption that some points are more informative then others. It tries to find global informative points and then assigns a weight to each of the points in training data based on their informativeness. It then uses weighted euclidean metric to calculate distances.

In another recent work [26], a k Exemplar-based Nearest Neighbor (kENN) classifier was proposed which is more sensitive to the minority class. The main idea is to first identify the exemplar minority class instances in the training data and then generalize them to Gaussian balls as concept for the minority class. The approach is based on extending the decision boundary for the minority class.

2.3 Discussion

The current existing regression algorithms [33, 10, 11, 4, 19] suffer from one or more of high computational complexity, poor results, fine tuning of parameters and extensive memory requirements. Finding an optimal decision tree [4] is a NP-Complete problem [22]. Neural networks are highly dependent on the initialization of weight vectors and generally, have large training time. Also, the best fit structure of the neural network has to be intelligently guessed or determined by trial and error method. However kNN provides excellent accuracy, but has linear computational complexity. This motivated us to strive for a kNN based regression algorithm which has a significantly low run time.

Learning from Imbalanced data has been figured out as a challenging problem in data mining. In the recent past, a lot of research work centered at kNN methodology has been done that try to improve its performance over imbalanced data and, significant progress is achieved. We also propose a novel class based wighted kNN algorithm, that is specially tuned for imbalanced data.

Chapter 3

BINER : BINary search based Efficient Regression

In this chapter, we present the BINER algorithm. In Section 3.1, we introduce the intuition and methodology of BINER followed by the actual algorithm in Section 3.2. In section 3.2.1, we present the computational complexity of the algorithm. Next, we describe the performance model in Section 3.3.1, followed by experimental results in Section 3.3.2 and discussion of the results in Section 3.3.3. Finally in Section 3.4, conclusion are drawn.

3.1 Intuition and Methodology of BINER

BINER follows the same overall methodology as kNN [18]. In a nutshell, kNN follows this approach:

1. It finds the k nearest neighbors to the given query.

2. Weighted mean of response variables in k nearest neighbors is given as output. The weights are kept inversely proportional to distance from the query.

The intuition of BINER is that the query Q is expected to be similar to tuples whose response variable values are close to that of Q. Thus it is more beneficial to find nearest neighbors in a *locality* where tuple have nearby response variable values rather than the whole dataset. This guarantees that even if the tuples in the considered locality are not the global nearest neighbors (nearest neighbors of the query in the complete dataset), the value of predicted response variable will be more appropriate.

Thus, the approach boils down to determining the locality (of nearby response variable) in which to conduct the nearest neighbor search. Once the locality is determined, response variable can be estimated as a weighted mean of responses of k nearest neighbors in this locality where weight is inversely proportional to the distance from the query.

Like other kNN based approaches, BINER has the following core assumption - tuples with similar X-values have similar response variable values. This assumption is almost always borne out in practice and is justified also by our experiments.

3.2 The BINER Algorithm

The algorithm proceeds in two steps.

1. It finds the range of tuples where the query Q has the maximum likelihood of occurrence. The term range (or locality), here, refers to consecutively indexed tuples in the dataset D and thus is characterized by two integers namely, start index and end index.

2. kNN is applied to these few (compared to D) tuples, and weighted mean of the K nearest neighbors in these ranges is quoted as output.

To find the range in which the query has the maximum likelihood of occurrence, the dataset is sorted in, say, non-decreasing manner of response variable values and then the function $biner$ described below is invoked with Q as query, and range $(0, n)$ where n is the number of tuples in D.

Algorithm 1 BINER : Pseudo code

Input: $Query\ instance\ Q, Range\ (start,\ end)$
Output: $Range\ (s, e)$
1: **while** $end - start > 2 * K$ **do**
2: $r = e - s$
3: $s_1, e_1 = start, start + r/2$
4: $s_2, e_2 = start + r/4, start + 3r/4$
5: $s_3, e_3 = start + r/2, end$
6: $d_1 = getDistance(RangeMean(s_1, e_1), Q)$
7: $d_2 = getDistance(RangeMean(s_2, e_2), Q)$
8: $d_3 = getDistance(RangeMean(s_3, e_3), Q)$
9: **if** $similar(d_1, d_2, d_3)$ **then**
10: **return** $(start,\ end)$
11: **else if** $similar(d_1, d_2, d_3)$ **then**
12: $start,\ end\ =\ s_i, e_i$
13: **end if**
14: **end while**
15: **return** $(start,\ end)$

The function, $biner$, iteratively bisects the current range until a range with size less than or equal to $2 * K$ is obtained (line 1) or a confident decision of bisecting a range cannot be taken (line 9-10), explained below. For each range, it makes three choices (lines 2-5) of half sized subranges namely, the lower half subrange (s_1, e_1), the center floating half (s_2, e_2) and the upper half (s_3, e_3), and computes distance of the query from these ranges (lines 6-9). The second subrange i.e. (s_2, e_2) is made to overlap with the other two ranges so as to ensure that tuples at end of first range and at start of third range get their due importance.

The distance of query Q from a range is calculated as

$$\sqrt{\sum \frac{(\mu_i - q_i)^2}{\sigma_i^2}} \tag{3.1}$$

where q_i is the ith attribute of the query, μ_i is the mean of ith attribute values in all tuples in the range and σ_i is the standard deviation of values of the ith attribute in the *whole* dataset, D. Standard deviation shows how much variation there is from the mean, μ. A low standard deviation indicates that the data points tend to be very close to the mean whereas high standard deviation indicates that the data points are spread out over a large range of values. Thus standard deviation helps in understanding how good a representational point the mean is for the range. Hence, the term σ^2 occurs in the distance metric.

In order to find the subrange in which the query has the maximum likelihood of occurrence, the distance of subranges from the query are compared. Among the three subranges, the query has the maximum likelihood of occurrence in the subrange which has minimum distance from the query. Thus the subrange with minimum distance from the query is considered and the complete process is repeated again.

When the size of range becomes small, the distances of subranges from the query tend to have same or close by values. In such situations, subranges have similar tuples and thus their distances become *similar*. Hence, a confident decision to select a subrange cannot be made and the current range is returned (lines 9-10). It may be noted that only the two minimum distances are checked for similarity. If the two larger distances are similar, a confident decision of selecting the subrange with minimum distance can be made.

We say that two distances, d_i and d_j are *similar* if $\min(d_i/d_j, d_j/d_i)$ is greater than 0.95. The value of 0.95 was selected by experimentations and it works well on most of the datasets as shown in the experimental section. The limiting size of $2 * K$ was chosen in order to keep a margin for selection of K nearest neighbors.

The above process obtains a range (or locality) where the query point has maximum likelihood of occurrence. Then kNN is applied on the range (locality) returned by $biner$ and weighted mean of response variables in K nearest neighbors is quoted as output. The weights are taken as $1/dist$, where $dist$ is the distance of the query and tuple.

3.2.1 Complexity Analysis

Before presenting the complexity analysis, we would like to first mention the preprocessing done here.

1. The first step is to sort the dataset D in, say, non-decreasing manner of the response variable.

2. Standard deviation, σ_i, of each attribute x_i is calculated and stored.

3. We store the mean, of all data points in all the feasible ranges (or localities) that may be encountered in our algorithm, in a Hash Table. Hence, calculation of mean becomes a constant order step. The key for the Hash Table, thus, will be two integers denoting the start and end index of tuples of in the range.

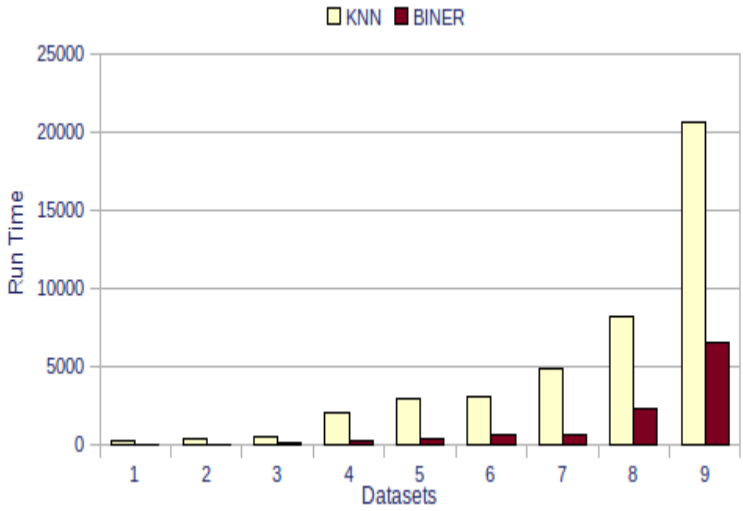

Figure 3.1 Comparison of run times of kNN and BINER

All feasible subranges are formed by recursively dividing a range into 3 ranges and stopping the recursion, when the size of range becomes less than or equal to $2 * K$. The value of K, is a parameter and is the same the one that would had been set for kNN. The range with start and end indices s and e respectively, is divided into 3 ranges namely [s, $s + (e - s)/2$], [$s + (e - s)/4$, $s + 3 * (e - s)/4$] and [$s + (e - s)/2$, e].

The algorithm at each stage divides the current range into 3 subranges each of half size of current range and considers one of them for subsequent processing. It can be observed that the function iterates $O(\log n)$ time. The function returns a range of size, say, R which is significantly smaller than n as confirmed by our experimentations. Thus computational complexity of the algorithm becomes $O(\log n+R)$ and when $R << n$ it becomes logarithmic. We illustrate our run time analysis on 7 datasets in Fig. 3.1.

3.3 Experimental Study

3.3.1 Performance Model

In this section, we demonstrate our experimental results. The experiments were done on a wide variety of datasets obtained from UCI data repository [1], Weka Datasets [1] and ML data repository [2]. A short description of all the datasets used is provided in Table 3.1. We have evaluated our results against the standard existing state of art approaches. The algorithms used were available in Weka toolkit [16]. All the results have been obtained using 10-fold cross validation technique.

[1]Weka Dataset Repository: http://www.cs.waikato.ac.nz/ml/weka/index_datasets.html
[2]ML Dataset Repository, http://mldata.org/repository/data/

Table 3.1 Dataset Description

Dataset	#Instances	#Attributes
Autompg	392	8
Bank	8192	9
Bodyfat	252	15
Concrete	1030	9
Forestfire	517	11
Flow	103	8
Housing	507	14
Space	3107	7
Slump	103	8
Synfriedman	500	6
Synfriedman 1	500	6
Synfriedman 2	500	6

We have used two metrics for quantifying our results, namely, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). MAE is mean of the absolute errors (actual output - predicted output). RMSE is square root of mean of squared errors.

Given a regressor R and an input tuple t_i , let the value of the output variable be y_i and the value output by the regressor be $R(t_i)$. Thus, the error made by the regressor for the given sample is: $absolute(R(t_i) - y_i)$. If the test set consists of m test samples (t_1 , . . . , t_m) then the Mean Absolute Error (MAE) is given as:

$$MAE = \frac{\sum_{i=0}^{m} absolute(R(t_i) - y_i)}{m} \tag{3.2}$$

and the root mean square error (RMSE) is given as:

$$RMSE = \sqrt{\frac{\sum_{i=0}^{m} (R(t_i) - y_i)^2}{m}} \tag{3.3}$$

We compared our performance against the following approaches: K Nearest Neighbor (Inverse distance Weighted) , Isotonic, Linear Regression, Least Mean Square (LMS) algorithm, Radial Basis Function Network (RBF Network), Regression Tree (RepTree) and Decision Stump.

3.3.2 Results

Table 3.2 and Table 3.3 compare the result of our algorithm with other existing state of art approaches.

Dataset	Autompg		Bodyfat		Flow		Housing		Space		Synfriedman2	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
BINER	**1.67**	**2.28**	**0.41**	**0.50**	**10.54**	**14.43**	**2.33**	**2.92**	**0.10**	**0.17**	52.55	64.35
IV-kNN	**2.40**	**3.59**	0.49	0.62	11.89	16.42	4.14	5.52	**0.10**	**0.17**	**48.64**	**60.65**
Isotonic	3.16	4.30	0.52	0.67	11.55	14.18	3.80	5.32	0.12	0.16	219.09	284.46
Linear Reg	2.56	3.40	**0.43**	**0.56**	**10.99**	**13.26**	**3.39**	**4.91**	0.11	0.16	108.44	145.69
LMS	**2.50**	**2.59**	0.45	0.58	13.28	18.56	3.42	5.55	0.11	0.15	109.62	153.43
RBF Network	3.90	5.07	0.61	0.77	14.76	17.42	6.13	8.42	0.14	0.19	246.71	317.76
Rep Tree	**2.30**	**3.31**	0.52	0.67	12.11	15.57	3.18	4.84	**0.10**	**0.14**	**45.34**	**61.10**
Decision Stump	4.20	5.18	0.63	0.80	12.48	15.41	5.61	7.50	0.13	0.18	256.93	320.71

Table 3.2 Comparison of results of BINER and other standard approaches

Dataset	Bank		Concrete		Forestfire		Slump		Synfriedman1		Synfriedman	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
BINER	**0.02**	**0.03**	**5.41**	**8.10**	**13.67**	**25.74**	6.01	9.79	**1.77**	**2.23**	**1.58**	**2.05**
IV-kNN	**0.02**	**0.03**	**4.15**	**6.10**	14.69	24.37	**5.89**	**9.83**	**1.75**	**2.16**	**1.53**	**2.03**
Isotonic	0.03	0.46	10.81	13.45	19.18	63.50	**5.86**	**7.52**	3.59	4.32	3.69	4.44
Linear Reg	**0.02**	**0.03**	8.30	10.45	19.92	64.28	6.67	7.82	2.76	4.65	2.25	2.83
LMS	0.03	0.05	9.52	17.53	**12.88**	**64.91**	6.68	10.56	2.45	4.71	2.24	2.82
RBF Network	0.04	0.06	13.38	16.56	18.86	63.86	6.98	8.73	3.92	4.91	3.86	4.81
Rep Tree	**0.02**	**0.03**	**5.43**	**7.38**	19.24	64.56	6.14	8.19	2.57	3.24	2.69	3.45
Decision Stump	0.04	0.05	11.54	14.46	18.93	64.68	7.05	8.86	3.69	4.40	3.73	4.63

Table 3.3 Comparison of results of BINER and other standard approaches

3.3.3 Discussion

We discuss our results in this section. It can be seen that our algorithm outperforms other regression algorithms in most of the datasets. Also it is clear from the results that BINER provides competitive results and is yet faster then kNN which is the current state of art.

3.4 Conclusions

In this chapter we have presented a new regression algorithm and evaluated it against existing standard regression algorithms. Our work is focused on finding a small locality in which the k nearest neighbors of query points have the maximum likelihood of occurrence. In addition to this, it allows users to use a (computationally) complex distance metric without significant increase in run time. We showed that the algorithm, more than often, outperforms existing standard state of art approaches on a wide variety of datasets and is faster by an order of magnitude.

Chapter 4

CLUEKR : CLUstering based Efficient kNN Regression

In this chapter, we present the CLUEKR algorithm for both regression and classification task. In Section 4.1, we introduce the CLUEKR algorithm for regression, followed by description of pre-processing step in Section 4.1.1 and actual algorithm in Section 4.1.2. In Section 4.1.3, we present the computational complexity of the algorithm. Next, we describe the performance model in Section 4.2.1, followed by experimental results and discussion in Section 4.2.2. Section 4.3 deals with modifying CLUEKR for classification task. Finally in Section 4.4, conclusion are drawn.

4.1 The CLUEKR Algorithm

We describe our proposed algorithm in this section. Our algorithm proceeds in two steps :

- It first finds the cluster in the hierarchy, in which the query point has maximum likelihood of occurrence.
- kNN is applied to points present in the cluster, and weighted mean of the k nearest neighbors of query point in the cluster is quoted as output.

Size of the obtained cluster is less compared to the size of entire dataset, in this way our algorithm reduce the search space for K-Nearest Neighbor algorithm. Now we explain in detail about the clustering phase (pre-processing step) first and later throw light on the actual algorithm.

While CLUEKR shares the overall methodology of BINER (described in previous chapter) it has major advantage:(1) It can be applied for classification and (2) As it uses clustering while splitting the data nodes, it ensures that closer points are put together and dissimilar points are separated neatly.

4.1.1 Pre-Processing

In the pre-processing step, data is hierarchically clustered and mean value for each of the cluster is calculated and stored. Each node in this hierarchy consist of clusters containing data point, which are recursively divided into three child clusters as we move down the hierarchy. We make use of the

fact that similar instances have similar value of response variable (basic analogy on which kNN works), while selecting the cluster center. Each cluster node is sorted based on the value of response variable, then $n/4$ and $3n/4$ ranked instance are selected as the center for two of the child clusters. For the third cluster, mean of the other two cluster's center is taken as center. These center are called as initial center. All the points present in the cluster node are divided into child cluster, based on to which child cluster center the point is closest.

We aim to divide each cluster nodes is such a fashion, that each child node contains half the data present in the parent node. However at each level we have added an extra child cluster to make the division of points smooth, this cluster also contain points belonging to other cluster that lies at its boundary with other cluster. This will help to properly classify the query instance that lie at the boundary of clusters. Boundary points to third cluster are defined as, points whose distance ratio from other cluster center to third cluster center is between 0.9 to 1.0.

We recursively divide the cluster nodes, till we get a cluster node which contain less that $2 * k$ points. The limiting size of $2 * k$ was chosen in order to keep a margin for selection of k nearest neighbors.

4.1.2 Actual Algorithm

Pseudo code for the actual algorithm is provided in Algo. 4. The recursive search starts from the root node and goes down the hierarchy. In order to find the cluster among the child nodes (for current node in the search) in which the query point has maximum likelihood of occurrence, distance of query point from mean value of all the child cluster of current node is calculated (lines 1-3). If the two closest distances comes out to be similar ($Confidance$ returns $false$) then we can't say with confidence, that which child cluster to pick and hence kNN algorithm is applied to the current cluster (line 5), else recursive search continues on the child cluster which is closest to the Query point (lines 7-8) i.e distance from mean of that cluster is least.

We say that two distances, d_i and d_j are similar if $min(d_i/d_j, d_j/d_i)$ is greater than 0.90. The value of 0.90 was selected by experimentations and it works well on most of the datasets as shown in the experimental section.

4.1.3 Complexity Analysis

We perform complexity analysis for both pre-processing step and actual algorithm in this section.

- In the pre-processing step, hierarchy consisting of cluster node is constructed, for which each cluster node is divided into child clusters. Assuming that we have data points sorted based on the value of response variable in the cluster node to be split, we can select the centers for child cluster directly and then data can be redistributed among the clusters in $O(n_s)$, where n_s is the size of cluster node to be spitted. If the cluster node is transversed in sorted order of response variable to redistribute data in child cluster, then child cluster will also contain data, sorted based on response variable. So if the root node has sorted data based on response variable, all other nodes

Algorithm 2 CLUEKR : Pseudo code

Input: $Query\ instance\ Q, Pointer\ to\ Root\ node\ in\ hierarchy\ R, Parameter\ k$
Output: $Value\ of\ Response\ Variable\ for\ Query\ instance\ Q$

1: **for** $each\ node\ j\ in\ childs\ of\ R$ **do**
2: $dist[j] = distance(Q, mean(j))$
3: **end for**
4: **if** $Confidance(dist)$ **then**
5: $ChildPtr = argmin(dist)$
6: **return** $CLUEKR(Q, ChildPtr,\ k)$
7: **else**
8: **return** $\mathrm{k}NN(Q,\ Data\ of\ R,\ k)$
9: **end if**

in the hierarchy will also follow the same ordering of data. As data is distributed from parent to child cluster, total amount of data present in child cluster is equal to data present in the parent cluster (ignoring the extra boundary points present in third cluster). Total time complexity of pre-processing step comes out to $O(n * log(n)) + O(h * n)$, where $n * log(n)$ is the cost involved in sorting root node and $h * n$ is the cost of splitting cluster nodes to construct a hierarchy of height h. This also involve the cost of calculating mean of each cluster node, as it can be calculated during the transversal of cluster node for distribution of data.

- At runtime our algorithm first performs a search for the cluster in the hierarchy,in which the query point has maximum likelihood of occurrence and then kNN is applied to the obtained cluster. Runtime complexity of our algorithm is $O(h + n_c)$, where h is the height of the hierarchy and n_c is the size of the cluster obtained. As n_c is supposed to be less compared to n, our algorithm is faster at runtime compared to kNN.

4.2 Experimental Study

4.2.1 Performance Model

In this section, we demonstrate our experimental settings. The experiments were obtained on a wide variety of real life datasets obtained from UCI data repository [1] and Weka Datasets [16]. A short description of all the datasets used is provided in Table 4.1. We have compared our performance against the following approaches: K Nearest Neighbor, Isotonic , Linear Regression(Linear Reg.), Least Mean Square (LMS) algorithm, Radial Basis Function Network (RBF Network), Regression Tree (RepTree) and Decision Stump(Dec Stump). Most of the algorithms are available as part of the Weka toolkit. All the results have been obtained using 10-fold cross validation technique.

Table 4.1 Dataset Description

Dataset	#Instances	#Attributes
Autompg	392	8
Bank	8192	9
Bodyfat	252	15
Concrete	1030	9
Cpu	209	7
Forestfire	517	11
Flow	103	8
Housing	507	14
Space	3107	7
Slump	103	8
Synfriedman	500	6
Synfriedman 1	500	6

We have used two metrics for quantifying our results, namely, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE)[1]. MAE is mean of the absolute errors (actual output - predicted output). RMSE is square root of mean of squared errors. We have used euclidean distance matrix to calculate the distances in our algorithm.

4.2.2 Results and Discussion

Table 4.2 compares the results obtained for our algorithm with other existing state of art approaches, top two results are highlighted in bold. As can be seen from the results, for majority of the datasets, we perform as accurate as kNN, however for some of the datasets, our algorithm also outperform kNN. Also our algorithm more than often outperforms other existing state-of-the art algorithms. However for concrete dataset in which the response variable is highly non-linear function of its attributes, we do not perform as good as for other datasets.

Table 4.3 compares average size of cluster obtained by our algorithm with original dataset size. Ratio column in the table shows the percentage ratio of the cluster size obtained by our algorithm to the original dataset size. It is clear from the data, that our algorithm reduces the search space for kNN significantly, however the degree of reduction varies depending on the type of dataset.

4.3 CLUEKR for Classification Task

4.3.1 Algorithm

Classification task requires to predict the class variable for a query point, which is a categorical attribute. On the other hand regression analysis helps to predicts the value of response variable for a

[1]Described in Section 3.3 of Chapter 3

Dataset	CLUEKR		*k*NN		Isotonic		Linear Reg.		LMS		RBF		Rep Tree		Dec Stump	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Autompg	**2.36**	**3.24**	**2.40**	**3.59**	3.16	4.3	2.56	3.4	**2.50**	**2.59**	3.90	5.07	**2.30**	**3.31**	4.2	5.18
Bank	**0.02**	**0.03**	**0.02**	**0.03**	0.03	0.46	**0.02**	**0.03**	0.03	0.05	0.04	0.06	**0.02**	**0.03**	0.04	0.05
Bodyfat	0.51	0.65	0.49	0.62	0.52	0.67	**0.43**	**0.56**	**0.45**	**0.58**	0.61	0.77	0.52	0.67	0.63	0.8
Concrete	7.20	9.92	**5.71**	**8.14**	10.81	13.45	8.30	10.45	9.52	17.53	13.38	16.56	**5.43**	**7.38**	11.54	14.46
Cpu	**21.45**	**70.76**	**18.79**	**74.37**	**23.12**	**50.95**	36.05	66.24	33.97	108.32	51.87	126.35	32.34	93.21	71.56	126.40
Flow	**11.27**	**14.78**	11.89	16.42	11.55	14.18	**10.99**	**13.26**	13.28	18.56	14.76	17.42	12.11	15.57	12.48	15.41
Forestfire	21.83	69.04	21.75	68.13	19.18	63.5	19.92	64.28	**12.88**	**64.91**	**18.86**	**63.86**	19.24	64.56	18.93	64.68
Housing	**2.96**	**4.63**	**2.97**	**4.63**	3.80	5.32	3.39	4.91	3.42	5.55	6.13	8.42	3.18	4.84	5.61	7.5
Slump	**5.56**	**7.75**	5.89	9.83	**5.86**	**7.52**	6.67	7.82	6.68	10.56	6.98	8.73	6.14	8.19	7.05	8.86
Space	**0.10**	**0.17**	**0.10**	**0.17**	0.12	0.16	0.11	0.16	0.11	0.15	0.14	0.19	**0.10**	**0.14**	0.13	0.18
Synf.	**2.10**	**2.64**	**1.96**	**2.45**	3.69	4.44	2.25	2.83	2.24	2.82	3.86	4.81	2.69	3.45	3.73	4.63
Synf. 1	**1.92**	**2.48**	**1.75**	**2.16**	3.59	4.32	2.76	4.65	2.45	4.71	3.92	4.91	2.57	3.24	3.69	4.4

Table 4.2 Comparison of results of CLUEKR with other standard approaches

Table 4.3 Comparison of dataset size with size of the cluster obtained

Dataset	Dataset Size	Cluster Size	%Ratio
Autompg	392	76	20
Bank	8192	2975	36
Bodyfat	252	50	20
Concrete	1030	191	18
Cpu	209	53	25
Forestfire	517	156	30
Flow	103	35	33
Housing	507	105	20
Space	3107	262	9
Slump	103	34	33
Synfriedman	500	143	28
Synfriedman 1	500	136	27

Table 4.4 Dataset Description

Dataset	#Instances	#Attributes	#Class
Cmc	1473	10	3
Diabetes	768	9	2
Heart	270	14	2
Hungarian	294	14	2
Ionosphere	351	34	2
Iris	150	4	3
Tranfusion	748	5	2
Wine	178	13	3

query point, which is a numeric attribute. In one of the steps of CLUKER, we need to sort the testing data based on the value of response variable, however this step cant be carried out for classification task as the attribute is not numeric. If we look carefully the sorting step is only needed to select the initial center points. If we can come up with a way to select the initial center points for classification data, then we can readily apply CLUEKR for classification task.

In this section we propose a way to modify CLUEKR for classification task. While distributing the data of parent node into child nodes, select two random points from the parent node as initial center for child clusters. The mean of these two initial center will serve as the center for the third child cluster. After this we iterate to choose child clusters center as the mean of the data present in the cluster. The actual search algorithm will remain the same only pre processing step will change.

4.3.2 Experimental Study

4.3.2.1 Performance Model

In this section, we demonstrate our experimental settings. The experiments were obtained on a wide variety of real life datasets obtained from UCI data repository [1] and Weka Datasets [16]. A short description of all the datasets used is provided in Table 4.4. We have compared our performance against the K Nearest Neighbor algorithm. All the results have been obtained using 10-fold cross validation technique.

4.3.2.2 Results and Discussion

Table 4.5 compares the results obtained for our algorithm with Regular K-Nearest Neighbor Algorithm. As can be seen from the results, for majority of the datasets, we perform nearly as accurate as kNN. Ratio column in the table shows the percentage ratio of the cluster size obtained by our algorithm to the original dataset size. It is clear from the data, that our algorithm reduces the search space for kNN significantly, however the degree of reduction varies depending on the type of dataset.

Table 4.5 Comparison of results of CLUEKR (for classification task) with kNN

Dataset	Our Accuracy	Regular kNN Accuracy	Dataset Size	Cluster Size	%Ratio
Cmc	48.35	50.65	1473	198	14
Diabetes	75.39	75.78	768	146	19
Heart	82.96	83.70	270	82	30
Hungarian	84.35	85.03	294	74	25
Ionosphere	84.33	83.76	351	71	20
Iris	94.67	96.67	150	51	34
Tranfusion	79.27	78.47	748	70	09
Wine	95.05	95.05	178	28	16

4.4 Conclusion

In this chapter, we have proposed a novel clustering based k nearest neighbor regression algorithm which is efficient and accurate. Our work is based on reducing the search space for nearest neighbors for any given point. We hierarchically cluster the data in pre-processing step and then search is performed to find the cluster in the hierarchy, in which the query point has the maximum likelihood of occurrence. We have also evaluated our approach against the existing state-of-the-art regression algorithms. As shown in the experimental section, our approaches reduces the search space for kNN significantly and is yet accurate. We have also proposed some modification to CLUEKR so that it can be applied for classification task.

Chapter 5

Class Based Weighted K-Nearest Neighbor Over Imbalance Dataset

In this chapter, we propose a Class based Weighted K-Nearest Neighbor Algorithm. In Section 5.1, we present an experimental study that deals with the design of Weighting factor for K-Nearest Neighbor Algorithm. We propose our modified algorithm in Section 5.2, followed by description about the property of weighting factor in Section 5.2.1. In Section 5.2.2, we present the computational complexity of the algorithm. Next, we describe the performance model in Section 5.3.1, followed by experimental results and discussion in Section 5.3.2 and 5.3.3 respectively. Finally in Section 5.4, conclusion are drawn.

5.1 An Experimental Study dealing with the design of Weighting Factor

In this section, we will present an experimental study that will throw some light on how the class based weighting factor for kNN classifier should be designed, so that it takes into account the nature of the data during classification.

Fig. 5.1 shows an artificial two-class imbalance problem, where the majority class "A" is represented by circle and the minority class "B" by triangle. The query instance is represented by cross. As can be seen from the figure, the query instance would have been classified as the majority class "A" by regular kNN algorithm with k value equal to 7. But if the classifier had taken into account the imbalance nature of the data then it would have classified the query instance as belonging to minority class "B".

To tune the existing kNN classifier so that it takes into account the nature of the data, we propose and evaluate three different Class based Weighting Factors for kNN classifier.

5.1.1 Design 1

In this section we introduce a simple base design of class based weighting factor for kNN classifier, which can be formally expressed as follows:

$$W[c] = \frac{1}{frequency[c]} \tag{5.1}$$

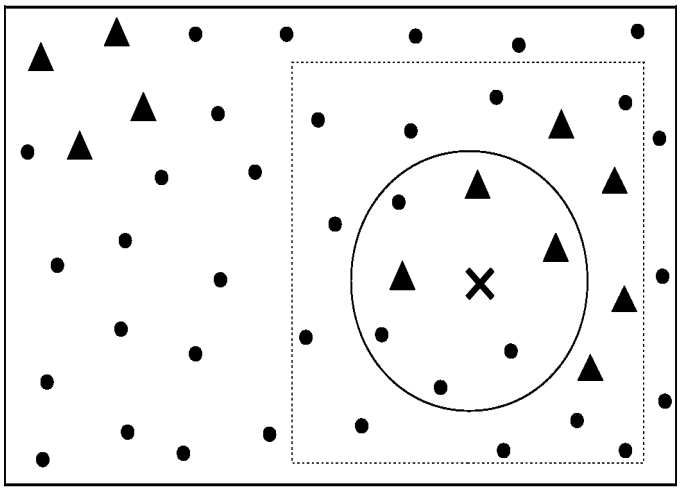

Figure 5.1 A sample scenario where regular kNN algorithm will fail

$W[c]$ denotes the weighting factor for class c. Also $frequency[c]$ here, denotes the frequency of occurrence of class c in the entire data. If the data is perfectly balanced, which implies $frequency[c]$ values for all the classes are approximately equal and hence the modified kNN algorithm (that uses this design of weighting factor) on balance data reduces to regular kNN algorithm. The modified kNN rule can be formally expressed as follows, for a given query instance x_t :

$$y_t = \underset{c \in \{c_1, c_2, ..., c_m\}}{\arg\max} \sum_{x_i \in N(x_t, k)} W[c] * E(y_i, c) \tag{5.2}$$

This Weighting factor has major drawbacks and does not perform equally well on majority of the datasets, as evident from our experimental study. This design only takes into account the overall degree of imbalance among the classes but not the local class distribution. One of the scenarios where this design fails is, when the data points are clustered such that each cluster has one major class and class distribution is imbalanced. In this case, the algorithm tends to favor the minority class over others. This scenario is represented in Fig. 5.2.

The factor 1/(cardinality of class in the overall dataset) does not serve as a good approximation to capture the nature of the overall dataset, which calls to enhance the factor.

5.1.2 Design 2

As the earlier proposed design does not perform well on majority of the datasets and has major drawbacks, we need to tune it accordingly. In our new proposed design, we bring in a coefficient $alpha$, which is based on the imbalance nature of the dataset. So the modified kNN rule for a given query instance x_t can be formally expressed as:

$$y_t = \underset{c \in \{c_1, c_2, ..., c_m\}}{\arg\max} \sum_{x_i \in N(x_t, k)} W[c] * E(y_i, c) \tag{5.3}$$

Where $W[c]$ denotes the weighting factor for class c, which can be defined as

$$W[c] = \frac{1}{1 + alpha * (\frac{frequency[c]}{\sum_{i=1}^{m} frequency[c_j]})} \tag{5.4}$$

Where $alpha$ is an input parameter, which can be taken as input from the user or can be learned from the data. The term $(\frac{frequency[c]}{\sum_{i=1}^{m} frequency[c_j]})$ in Eq. 5.4, is the ratio of occurrence of class c in the entire data to the total number of instances in the entire data.

For perfectly balanced data the value of alpha should be 0, which will make all $W[c]$ values equal to 1 and hence the modified kNN algorithm on balanced data reduces to existing kNN algorithm. The performance of modified kNN classifier is highly sensitive to the value of alpha. The alpha factor tries to bring into account the nature of distribution of the data, which was ignored in the earlier design. However the present design also fails to capture the local class distribution around the query instance.

The weighting factor is bounded between 1 and $(\frac{1}{1+alpha*(\frac{frequency[c]}{\sum_{i=1}^{m} frequency[c_j]})})$.

5.1.3 Design 3

In our new design, instead of considering the nature of the entire data, we only consider the region around the neighborhood of the query instance. More clearly if k is the number of neighbors that are used by the existing kNN algorithm to decide the class of the query instance, then in this design we take into account the class distribution in $k + d$ nearest neighbors of the query instance. In our previous designs, too much distant instances also have an affect while classifying the query instance, but in this one, we try to limit the region, so that instances within that region can only affect the decision. So for a given query instance x_t the modified kNN algorithm can be formally expressed as follows:

$$y_t = \underset{c \in \{c_1, c_2, \ldots, c_m\}}{\arg\max} \sum_{x_i \in N(x_t, k)} W(c, x_t) * E(y_i, c) \tag{5.5}$$

Where $W(c, x_t)$ denotes the weighting factor for class c and instance x_t , and can be defined as:

$$W(c, x_t) = \frac{1}{\sum_{x_i \in N(x_t, k+d)} E(y_i, c)} \tag{5.6}$$

Where d is an input parameter, which can be taken as input from the user or can be learned from the data. In Eq. 5.6, the term $(\sum_{x_i \in N(x_t, k+d)} E(y_i, c))$ is the number of instances of class c present in the $(k + d)$ nearest neighbors of x_t . This algorithm will more closely monitor the nature of the data around the neighborhood of query instance.

In Fig. 5.2, data points are present in clusters with each cluster having 1 major class. If design 1 based classifier is used for classification on this type of data, it would have favored the minority class and have assigned minority class label to query instance 1. In this case, regular kNN algorithm would fail to classify the minority class instances present at the boundary of the clusters region (for example query instance 2). The current design will succeed to classify both of the query instances correctly.

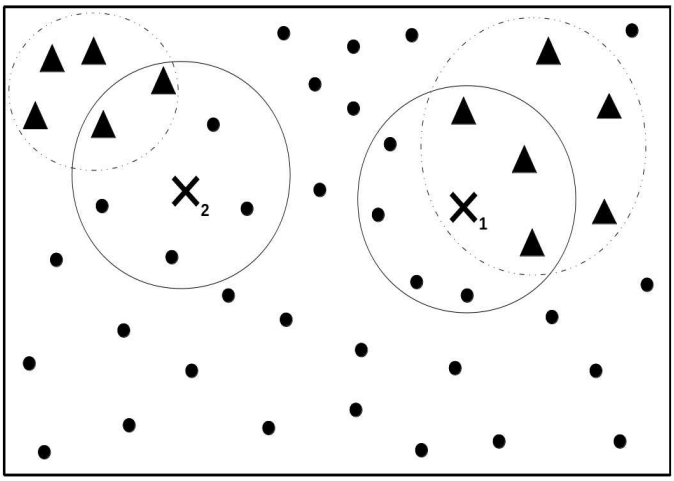

Figure 5.2 A sample scenario where data points are present in clusters with each cluster having 1 major class.

Weighting factor based on design 1 and 2, for each of the class is only a function of that class distribution, so it can be computed in the pre-processing phase. Hence it will computationally takes $O(n)$ time (where n is the total number of training instances present in the data) to calculate the weighting factor for all the classes in the pre-processing step. Hence the kNN classifier based on any of the two designs have the same runtime as existing kNN classifier. On the other hand for kNN classifier based on design 3, weighting factor for each of the class is a function of that class distribution and query instance hence cannot be calculated at the pre-processing stage. However for each query instance, weighting factor for all the classes can be calculated simultaneously while getting the neighbors of the query point. Hence kNN classifier based on design 3 is computationally a little more expensive then existing kNN classifier.

5.1.4 Experimental Study

5.1.4.1 Performance Model

In this section, we demonstrate results of our experimental study. The experiments were conducted on a wide variety of datasets obtained from UCI data repository [1], Weka Datasets [16]. A short description of all the datasets is provided in Table 5.1. All the results have been obtained using 10-fold cross validation technique. As it is known that the use of overall accuracy is not an appropriate evaluation measure for imbalanced datasets, because of the dominating effect of the majority class, we have used F-Score as the evaluation metric. F-Score considers both the precision and the recall of the test to compute the score.[1]

[1]We have used the average of F-score value obtained by taking each of the class present in the data as the positive class.

Table 5.1 Dataset Description

Dataset	#Instances	#Attributes	#Class	Minority Class%
Balance	625	5	3	7.84
Diabetes	768	9	2	34.9
Heart	270	14	2	44.44
Hepatitis	155	20	2	20.65
Ionosphere	351	34	2	35.9
Tranfusion	748	5	2	23.7

Table 5.2 Experimental results over several real world dataset

Dataset	Design 1	Design 2	Design 3	Regular kNN
Balance	0.819 (4)	0.908 (2)	0.912 (1)	0.865 (3)
Diabetes	0.745 (3)	0.751 (2)	0.757 (1)	0.740 (4)
Heart	0.822 (3)	0.843 (1)	0.843 (1)	0.840 (2)
Hepatits	0.836 (4)	0.862 (2)	0.865 (1)	0.856 (3)
Ionosphere	0.883 (2)	0.883 (2)	0.907 (1)	0.857 (3)
Tranfusion	0.723 (3)	0.781 (1)	0.781 (1)	0.777 (2)

For each type of kNN classifier, parameters values of that respective design are iterated over a defined set and maximum average F-Score value obtained is listed in the results table. Details of the iteration set for each of the parameters is as follows:

1. Value of k for each of type of classifiers is iterated from 1 to 30 at an interval 1, i.e 1, 2, 3, ..., 30.
2. Value of $alpha$ for classifier based on design 2, is iterated over the following set:
   (a) 0 to 1 at interval of 0.1, i.e. 0, 0.1, 0.2, ... ,1.
   (b) 2 to 10 at interval of 1, i.e. 1, 2, 3, ... ,10.
   (c) 15 to 100 at interval of 5, i.e. 0, 15, 20, ... ,100.
3. Value of d for design 3, is iterated from k to $sizeofthedataset$ at an interval of k, i.e k, $2 * k$, $3 * k$, ..., $a * k$.

5.1.4.2 Results

Table 5.2 compares the performance of our proposed kNN based classifiers with regular kNN on variety of real world datasets.

5.1.4.3 Discussion

We discuss the results obtained from our experimental study in this section. It can be seen that our proposed design 2 and 3 produce consistently accurate classifier in most of the datasets. Also classifier

based on design 2 and 3, always outperform regular kNN on all the datasets, this hints towards the fact that the modified kNN rule takes into account the nature of the data to classify it. Our experimental study also support the drawback of design 1 (mentioned in section 5.1.1), that it does not take into account the nature of the data and hence do not perform well on all the datasets.

5.1.5 Conclusion

In this section, we have proposed three new Class based Weighting factors for the K-Nearest Neighbor algorithm, so that it takes into account the nature of the data rather than simply ignoring it, as done in the current implementation of K-Nearest Neighbor algorithm. Our designs are focused on tuning the K-Nearest Neighbor for imbalance data, so that its performance on imbalance data is enhanced. Our experimental study hints towards designing a class based weighting factor for kNN based classifier that takes into account the local class distribution around the neighborhood of the query point. This type of weighting factor will help to improve the performance of kNN classifier over imbalance datasets.

5.2 Our Proposed Algorithm

In this section, we will explain in detail our proposed algorithm. To tune the existing kNN algorithm , we introduce a weighting factor for each class. Our algorithm can be formally expressed as follows, for a given query instance x_t:

$$y_t = \underset{c \in \{c_1, c_2, \ldots, c_m\}}{\arg\max} \sum_{x_i \in N(x_t, k)} W[c, x_t] * E(y_i, c) \tag{5.7}$$

Where $W[c, x_t]$ denotes the weighting factor for the class c, while classifying query instance x_t. For Weighting factor equal to 1 for all the classes, our algorithm reduces to the existing kNN classifier. This weighting factor is introduced to take into account, class distribution around the query instance. The proposed algorithm is sensitive to the value of weighting factor.

Now we discuss on how to learn the value of weighting factor for each of the classes. Fig. 5.3 illustrates an imbalance dataset, in which data points are present in clusters with each cluster having exactly one major class. In this case, regular kNN algorithm would fail to classify the minority class instances present at the boundary of the cluster region (for example query instance 1).

To design the weights, we considered both query dependent and query independent weighting factor. If our learned weighting factors have a constant value for each of the class through out the dataset i.e. they do not depend on the query instance, and favors the minority class then, our algorithm would have classified the minority class instances present at the boundary of the clusters correctly, but have not classified points like instance 2 correctly. Having only class dependent weighting factor values would not capture the data distribution around the neighborhood of the query instance.

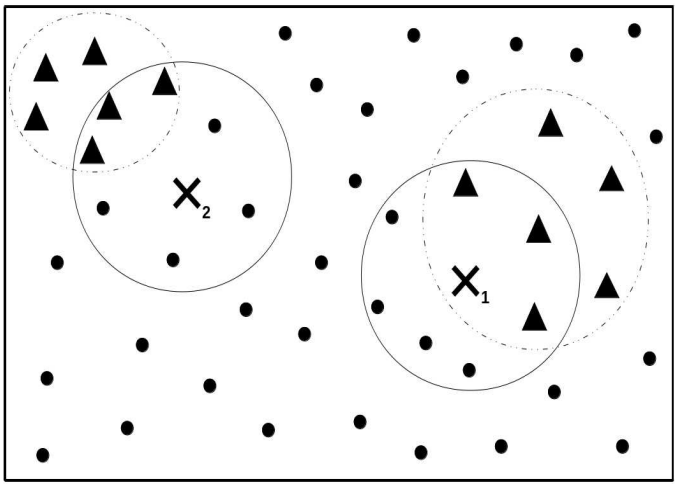

Figure 5.3 A sample scenario where data points are present in clusters (dotted circle represent clusters containing minority class) with each cluster having one major class.

Our weighting factor value $W[c, x_t]$ can be denoted as :

$$W[c, x_t] = \frac{alpha(c, x_t)}{1 + alpha(c, x_t)} \tag{5.8}$$

where

$$alpha(c, x_t) = \sum_{x_i \in N(x_t, \frac{k}{m}, c)} \frac{m * getcoef(x_i)}{k} \tag{5.9}$$

$$N(x, k, c) = Set\ of\ k\ nearest\ neighbor\ of\ x\ belonging\ to\ class\ c$$

$$getcoef(x_i) = \frac{N(x_i, k, c')}{N(x_i, k, y_i)} \tag{5.10}$$

where c' is the class to which x_i is classified by existing kNN classifier. if c' equals to y_i then $getcoef(x_i)$ turns out to be 1.

Hence for a query point the weighting factor of a class is calculated based on how the k/m nearest neighbors of query point belonging to that class are classified by the existing kNN classifier. If a instance is classified correctly $getcoef$ will return 1 for it, else it will return the value by which the class prior probability should be multiplied, so that it is classified correctly. The basic intuition about the above formulae is simple: "If instances of a specific class are poorly classified in a particular region, then that class is likely to be a minority class in that region and should be given a higher weight."

5.2.1 Properties of Weighting Factor

1. The weighting factor for each of the class is calculated based on how the k/m nearest neighbors of query point belonging to that class are classified by regular kNN classifier. If points belonging

Algorithm 3 : Pseudo code

Input: $Query\ instance\ Q, Training\ Data\ D, Parameter\ k,\ Set\ of\ all\ the\ class\ label\ C$
Output: $Class\ Label\ Cl\ for\ Query\ instance\ Q$

1: **for** $each\ class\ j\ in\ C$ **do**
2: $\quad Coefficient = (\frac{m}{k}) * \sum_{x_i \in N(Q, \frac{k}{m}, j)} (getcoef(x_i))$
3: $\quad W[j, Q] = \frac{Coefficient}{1+Coefficient}$
4: **end for**
5: **for** $each\ neighbor\ n\ in\ N(Q, k)$ **do**
6: $\quad Probability[class(n)] = Probability[class(n)] + W[class(n), Q]$
7: **end for**
8: $Cl = \arg\max Probability$

to a particular class in the neighborhood of the query points are classified incorrectly by regular kNN approach, then the weighting factor of that class will have a high value indicating that this class might be a minority class in that region. Hence the learned weighting factor takes into account the class distribution around neighborhood of the query point.

2. Value of weighting factor is bounded between 0.5 to 1, proof of which is :

   if $x_i\ is\ classified\ correctly\ by\ regular\ kNN$ **then**
   $\Rightarrow getcoef(x_i) \leftarrow 1$
   else $x_i\ is\ classified\ as\ belonging\ to\ class\ c'$
   $\Rightarrow N(x_i, k, c') > N(x_i, k, y_i)$
   $\Rightarrow getcoef(x_i) > 1$ (from Eq. 5.10)
   end if

   Hence $getcoef(x_i)$ value is always greater than or equal to 1, that implies $alpha(c, x_t)$ which is average of $getcoef$ values of k/m nearest neighbors of x_t is always greater than or equal to 1. Eq. 5.8 can also be written as

$$W[c, x_t] = \frac{1}{1 + \frac{1}{alpha(c, x_t)}}$$

   Which impliles that

$$0.5 \leq W[c, x_t] \leq 1$$

   If some outlier point is present in the neighborhood of the query point its $getcoef$ factor would have a high value, but as the weighting factor for a class is calculated by taking average of $getcoef$ values of k/m nearest neighbors of query point belonging to that class, its value would not be much affected by a outlier point. This makes our learned weighting factors resistant to outliers.

5.2.2 Complexity Analysis

The proposed algorithm needs to search for the k nearest neighbors of the query point (global nearest neighbors, line 5 of Algorithm 1.), same as the regular kNN algorithm. Apart from finding the global nearest neighbors, it also need to calculate the weighting factor for each of the class. Following calculations are involved in the calculation of weighting factors :

1. For each of the class, find k/m nearest neighbors of query point among that class (class neighbors). We can make use of the fact that the global k nearest neighbors will be among the k nearest neighbors for each of the class, to optimize the search. Rather then finding k/m nearest neighbor for each class, we first get the k nearest neighbor for each class and then get the global k nearest neighbors, with some extra cost involved. For example, assuming that no index structure is present in the training data, while searching for global k nearest neighbors of query point a binary heap structure of size k is needed. For our proposed approach, we maintain such a heap structure one per class. While searching for neighbors, each of the points in the training data is inserted in the heap belonging to its class. When all the points are inserted in the heap, we have the k nearest neighbors of query points among each class in the respective class heap. The total $heapinsert$ operations remains same as when finding global k nearest neighbors. Then we can find the global k nearest neighbors and class neighbors from this $k * (numberofclass)$ points.

2. For each of the points obtained above calculate $getcoef$ function, which needs the k nearest neighbors for each of the points (line 2). If $getcoef$ function is calculated for all the points present in the training data during the pre processing step then, this runtime overhead can be avoided. Else we need to find the k nearest neighbors of all the points obtained above i.e. k points.

5.3 Experimental Study

5.3.1 Performance Model

In this section, we demonstrate our experimental settings. The experiments were conducted on a wide variety of datasets obtained from UCI data repository [1] and Weka Datasets [16]. A short description of all the datasets is provided in Table 5.3. These datasets have been selected as they typically have a low minority class percentage and hence are imbalance. We have evaluated our algorithm against the existing state of art approaches. All the results have been obtained using 10-fold cross validation technique, except for SMOTE. For SMOTE each of the dataset is first randomized and then divided into training and testing data. SMOTE sampling is applied on training data to oversample the minority class and then regular kNN is used to classify instances present in testing data using the sampled training data.

Table 5.3 Dataset Description

Dataset	#Instances	#Attributes	#Class	Minority Class%
Balance	625	5	3	7.84
Cmc	1473	10	3	22.61
Pima	768	9	2	34.9
Glass	214	10	6	4.2
Heart	270	14	2	44.44
Hungarian	294	14	2	36.05
Ionosphere	351	34	2	35.9
Tranfusion	748	5	2	23.7
Wine	178	13	3	26.9

As it is known that the use of overall accuracy is not an appropriate evaluation measure for imbalanced datasets, because of the dominating effect of the majority class, we have used F-Score as the evaluation metric. F-Score considers both the precision and the recall of the test to compute the score. We compared our performance against the following approaches: Regular K Nearest Neighbors (kNN) [2], Exemplar kNN (kENN) [3], SMOTE , HDDT [4] , C4.5 , CCPDT [5], NaiveBayes (Naive). For all kNN based approaches (including SMOTE) F-Score obtained at maximum accuracy is mentioned.

5.3.2 Results and Discussion

Table 5.4 compares the result of our modified algorithm with existing state of the art algorithm. The number in the parenthesis indicates the rank of the respective algorithm. Also the top two algorithms are highlighted in bold. It can be seen that our approach produces consistently accurate classifier and outperforms other algorithms in most of the datasets. Also our proposed algorithm always outperform regular kNN on all the datasets, this confirms that the modified kNN algorithms takes into account the nature of the data to classify it. However for Ionosphere dataset decision tree based algorithms perform better than other state of the art algorithms.

Fig. 5.4 compares performance of our algorithm with kNN in terms of overall accuracy and accuracy to classify minority class, as the value of k varies for Hungarian dataset. It becomes clear from the figure that our algorithm based classifier are more sensitive to classify minority class and are still highly accurate. Also classifier learned from our approach are more accurate for larger values of k, this is evident from the fact that, for high value of k large region around the neighborhood of query point is considered to determine the local class distribution.

[2]For kNN , SMOTE, C4.5 (available as j48) and Naive, implementation available in Weka toolkit is used.
[3]The code is obtained from http://goanna.cs.rmit.edu.au/∼zhang/ENN/Weka-3-6_ENN.zip
[4]The code is obtained from http://nd.edu/∼dial/software/hddt.tar.gz
[5]The code is obtained from http://www.cs.usyd.edu.au/∼weiliu/CCPDT_src.zip

Table 5.4 Experimental results over several real world dataset

Dataset	Our Algo	kNN	kENN	SMOTE	HDDT	C4.5	CCPDT	Naive
Balance	**0.361 (1)**	0.077 (6)	**0.167 (2)**	0.154 (3)	0.089 (5)	0.000 (7)	0.092 (4)	0.000 (7)
Cmc	0.419 (3)	0.418 (4)	**0.424 (2)**	0.364 (7)	0.380 (6)	0.409 (5)	0.356 (8)	**0.445 (1)**
Pima	**0.628 (2)**	0.601 (6)	0.610 (5)	0.593 (7)	0.613 (4)	0.614 (3)	0.587 (8)	**0.643 (1)**
Glass	**0.778 (1)**	**0.778 (1)**	0.560 (7)	0.750 (3)	0.571 (6)	0.636 (5)	0.235 (8)	0.696 (4)
Heart	**0.812 (2)**	0.805 (5)	**0.812 (2)**	0.765 (7)	0.784 (6)	0.736 (8)	**0.828 (1)**	**0.812 (2)**
Hungarian	0.779 (3)	0.747 (6)	0.747 (6)	**0.805 (2)**	0.767 (4)	0.656 (8)	**0.815 (1)**	0.762 (5)
Ionosphere	0.824 (5)	0.779 (8)	0.793 (6)	0.833 (4)	**0.891 (2)**	0.874 (3)	**0.894 (1)**	0.781 (7)
Tranfusion	**0.489 (2)**	0.442 (7)	0.486 (4)	**0.509 (1)**	**0.489 (2)**	0.481 (6)	0.486 (4)	0.281 (8)
Wine	**0.980 (1)**	**0.980 (1)**	0.950 (6)	**0.980 (1)**	0.958 (5)	0.923 (7)	0.871 (8)	0.970 (4)
Average Rank	**2.22**	4.88	4.44	**3.88**	4.44	5.77	4.77	4.33

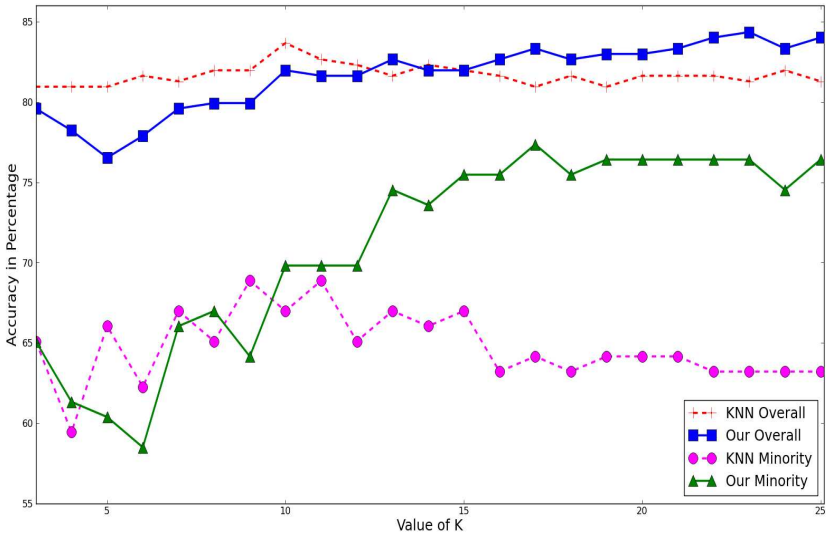

Figure 5.4 Performance Comparison between Our Algorithm and kNN

5.4 Conclusion

In this chapter, we have proposed a modified version of K-Nearest Neighbor algorithm so that it takes into account, class distribution around neighborhood of query instance during classification. In our modified algorithm, a weight is calculated for each class based on how its instances are classified by existing K-Nearest Neighbor classifier around the query instance and then a weighted kNN is applied. We have also evaluated our approach against the existing standard algorithms. Our work is focused on tuning the K-Nearest Neighbor for imbalance data, so that its performance on imbalance data is enhanced. As shown in the experimental section, our approach more than often, outperforms existing state of the art approaches on a wide variety of datasets. Also our modified algorithm perform as good as the existing K-Nearest Neighbor classifier on balance data.

Chapter 6

Integrating Class Based Weighted K-Nearest Neighbor approach with CLUEKR

In this chapter, we integrate Class based Weighted K-Nearest Neighbor approach with CLUEKR (for classification). In Section 6.1, we give a brief overview about the previous classifiers, then we talk about the integration steps involved in Section 6.2. Next, we describe the performance model in Section 6.3.1, followed by experimental results and discussion in Section 6.3.2. Finally in Section 6.4, conclusions are drawn.

6.1 Background

We have proposed two different kNN classifiers in the previous chapters, one having the advantage of low run time computational complexity and the other one is sensitive to local class distribution around the neighborhood of the query point. Now we will integrate the two classifiers to come up with an efficient *and* accurate kNN classifier that takes into account the local class distribution during classification. A short description about the two kNN classifiers is as follows:

1. CLUEKR (for classification): A novel, efficient and outlier resistant, clustering based kNN classifier, which instead of searching for nearest neighbors directly in the entire dataset, first find the cluster in which the query point should lie. Data is Hierarchically clustered in the pre processing step, then a recursive search starting from root node of the hierarchy is performed. At current search node in the hierarchy, we select a cluster among its child, in which the query point has maximum likelihood of occurrence and then a recursive search is applied to it. Finally we find the k nearest neighbors of query points in that cluster and return the majority class among the neighbors as the result.

2. Class Based Weighted kNN classifier: A Class based Weighted K-Nearest Neighbor algorithm in which a weight is assigned to each of the class based on how its instances are classified in the neighborhood of query instance by the regular kNN classifier. The modified algorithm takes into

account the local class distribution around the neighborhood of query instance. We ensure that the weights assigned do not give undue advantage to outliers.

6.2 Integration Step

In the pre-processing step $getcoef$ function is calculated for all the points present in the training data, based on how they are classified by regular kNN classifier.

In this integrated approach, CLUEKR algorithm in used to find the cluster node, where the query point has maximum likelihood of occurrence. Then a weighting factor for each of the class is calculated using ideas from the class based weighted kNN algorithm (described in previous chapter), based on how points belonging to that class are classified by regular kNN algorithm in the entire dataset. Then weighted kNN is applied and the predicted class is returned. As we have the $getcoef$ function value for all the points in the dataset, calculation of weighting factors can be done simultaneously while getting the k nearest neighbor of the query points in the cluster node (as discussed in section 5.3.2 of previous chapter).

We need to ensure that all the cluster nodes have enough points corresponding to each class, so that weighting factor for each of the class is calculated correctly. Hence if splitting a cluster node into child cluster results in a child node which don't have atleast k/m points belonging to each class, then we don't split.

Algorithm 4 Integrated Approach: Pseudo code

Input: $Query\ instance\ Q, Pointer\ to\ Root\ node\ in\ hierarchy\ R, Parameter\ k$
Output: $Predicted\ Class\ for\ Query\ instance\ Q$

1: **for** $each\ node\ j\ in\ childs\ of\ R$ **do**
2: $\quad dist[j] = distance(Q, mean(j))$
3: **end for**
4: **if** $Confidance(dist)$ **then**
5: $\quad ChildPtr = argmin(dist)$
6: $\quad$ **return** $CLUEKR(Q, ChildPtr,\ k)$
7: **else**
8: $\quad$ **return** $CWkNN(Q,\ Data\ of\ R,\ k)$
9: **end if**

Pseudo code for the integrated algorithm is provided in Algo. 4. The $CWkNN$ function first get the k/m nearest class neighbors of the query point in the cluster node (obtained by CLUEKR) for each of the class and then calculate the weighting factor using the $getcoef$ value of the points obtained (described in section 5.3.2 of previous chapter).

Table 6.1 Dataset Description

Dataset	#Instances	#Attributes	#Class	Minority Class%
Balance	625	5	3	7.84
Cmc	1473	10	3	22.61
Pima	768	9	2	34.9
Glass	214	10	6	4.2
Heart	270	14	2	44.44
Hungarian	294	14	2	36.05
Ionosphere	351	34	2	35.9
Tranfusion	748	5	2	23.7
Wine	178	13	3	26.9

6.3 Experimental Study

6.3.1 Performance Model

In this section, we demonstrate our experimental settings. The experiments were conducted on a wide variety of datasets obtained from UCI data repository [1] and Weka Datasets [16]. A short description of all the datasets is provided in Table 6.1. These datasets have been selected as they typically have a low minority class percentage and hence are imbalance. We have evaluated our algorithm performance against regular kNN and Class based Weighted kNN approach (as described in section 5). All the results have been obtained using 10-fold cross validation technique. we have used F-Score as the evaluation metric.

6.3.2 Results and Discussion

Table 6.2 compares the result of our integrated algorithm with other state-of-the-art classifiers (as described in section 5). It can be seen that our approach produces consistently accurate classifier and outperforms other algorithms in most of the datasets. Table 6.3 compares average size of cluster obtained by our integrated algorithm with original dataset size. Ratio column in the table shows the percentage ratio of the cluster size obtained by our algorithm to the original dataset size. It can be seen that the ratio of cluster size to original dataset has increased compared to CLUEKR, which is due to change in the pruning condition.

6.4 Conclusion

In this chapter, we have integrated Class Based Weighted K-Nearest Neighbor work with CLUEKR and present a efficient and accurate kNN based classifier that takes into account the nature of data. One of the major changes in the integration deals with the pruning step. As shown in the experimental sec-

Table 6.2 Experimental results over several real world dataset

Dataset	Integrated Algo	Our Algo	*k*NN	*k*ENN	SMOTE	HDDT	C4.5	CCPDT	Naive
Balance	**0.358 (2)**	**0.361 (1)**	0.077 (7)	0.167 (3)	0.154 (4)	0.089 (6)	0.000 (8)	0.092 (5)	0.000 (8)
Cmc	0.419 (3)	0.419 (3)	0.418 (5)	**0.424 (2)**	0.364 (8)	0.380 (7)	0.409 (6)	0.356 (8)	**0.445 (1)**
Pima	0.621 (3)	**0.628 (2)**	0.601 (7)	0.610 (6)	0.593 (8)	0.613 (5)	0.614 (4)	0.587 (9)	**0.643 (1)**
Heart	**0.812 (2)**	**0.812 (2)**	0.805 (6)	**0.812 (2)**	0.765 (8)	0.784 (7)	0.736 (9)	**0.828 (1)**	**0.812 (2)**
Hungarian	0.779 (3)	0.779 (3)	0.747 (7)	0.747 (7)	**0.805 (2)**	0.767 (5)	0.656 (9)	**0.815 (1)**	0.762 (6)
Ionosphere	0.819 (6)	0.824 (5)	0.779 (9)	0.793 (7)	0.833 (4)	**0.891 (2)**	0.874 (3)	**0.894 (1)**	0.781 (8)
Tranfusion	0.483 (6)	**0.489 (2)**	0.442 (8)	0.486 (4)	**0.509 (1)**	**0.489 (2)**	0.481 (7)	0.486 (4)	0.281 (9)
Wine	**0.980 (1)**	**0.980 (1)**	**0.980 (1)**	0.950 (7)	**0.980 (1)**	0.958 (6)	0.923 (8)	0.871 (9)	0.970 (5)

Table 6.3 Comparision of Final cluster size obtained with dataset size

Dataset	Dataset Size	Cluster Size	%Ratio
Balance	625	562	90
Cmc	1473	1125	76
Pima	768	280	37
Heart	270	135	50
Hungarian	294	294	100
Ionosphere	351	167	48
Tranfusion	748	134	18
Wine	178	178	100

tion, our integrated approach reduces the search space for Class Based Weighted K-Nearest Neighbor classifier significantly (on large datasets) and is yet accurate.

Chapter 7

Conclusions

In this thesis we have provided a suit of solutions to tackle some of the drawbacks of K-Nearest Neighbor algorithm. Two of our works, BINER and CLUEKR are focused on reducing the runtime of kNN. However CLUEKR can be applied for both regression and classification, while BINER only works for regression analysis. We have evaluated both of these algorithm against state-of-the-art regression algorithms on a variety of real world datasets. We have also proposed a Class based Weighted K-Nearest Neighbor algorithm which is focused on tuning kNN classifier so that it takes into account the local class distribution around the query point during classification. We have evaluated our modified algorithm against state-of-the-art classifiers on a variety of real world datasets. The datasets used have varying minority class percentages. Finally we have integrated Class based Weighted kNN approach with CLUEKR (for classification) to provide an accurate and efficient kNN classifier that takes into account local class distribution during classification.

We summarize the major contributions of the thesis in the Section 7.1 followed by directions for future work in Section 7.2

7.1 Contribution

The main contributions in this work are as the follows:

1. BINER : We contribute a new efficient technique for regression. Our algorithm instead of directly predicting the response variable, narrows down the range in which the response variable has the maximum likelihood of occurrence and then interpolates to give the output. The data is Hierarchically partitioned in the pre-processing step, and search for the partition in which the response has the maximum likelihood of occurrence is carried out at the runtime. It takes a single parameter k, the same as in kNN and more than often outperforms the conventional state of art methods on a wide variety of datasets as illustrated in the Experimental Section.

2. CLUEKR : We propose a novel, efficient and outlier resistant, clustering based kNN regression algorithm CLUEKR, which instead of searching for nearest neighbors directly in the entire dataset,

first find the cluster in which the query point should lie. We first Hierarchically cluster the data in the pre processing step, then a recursive search starting from root node of the hierarchy is performed. At current search node in the hierarchy, we select a cluster among its child, in which the query point has maximum likelihood of occurrence and then a recursive search is applied to it. Finally we find the k nearest neighbors of query points in that cluster and return the weighted mean of their response variable as result. We also propose the modification that enable CLUEKR to be applied for classification task. One of the major advantage of CLUEKR over BINER is that, with some small modification CLUEKR can be used for classification task.

3. Class Based Weighted K-Nearest Neighbor : We propose a modified K-Nearest Neighbor algorithm to solve the imbalanced dataset classification problem. More specifically the contributions are as follows:

   (a) First we present a mathematical model of K-Nearest Neighbor algorithm and show that, it does not take into account nature of the data around the query instance.

   (b) To solve the above problem, we propose a Weighted K-Nearest Neighbor algorithm in which a weight is assigned to each of the class based on how its instances are classified in the neighborhood of query instance by the regular K-Nearest Neighbor classifier. The modified algorithm takes into account class distribution around the neighborhood of query instance. We ensure that the weights assigned do not give undue advantage to outliers.

   (c) A thorough experimental study of the proposed approach over several real world dataset was performed. The study confirms that our approach performs better than the current state-of-the-art approaches.

4. Finally, we integrate Class Based Weighted K-Nearest Neighbor work with CLUEKR and present a efficient and accurate kNN based classifier that takes into account the nature of data.

7.2 Future Work

Future research directions for extending and improving the work done in this thesis are as follows:

1. In BINER algorithm, instead of simply bisecting at some fixed points, if we divide at points where there is some abrupt change in y values etc, probably our algorithm will work better. Such heuristic functions will further improve the performance of BINER.

2. In CLUEKR algorithm, at each level in the hierarchy, parent nodes are split into child clusters. If the clustering can be done in such a way that for any given query point, its k nearest neighbors in the entire dataset are same as its k nearest neighbors in the cluster node obtained by CLUEKR's search algorithm, then CLUKER will reduce to kNN. This will guarantee that its performance is theoretically the same as kNN.

3. Class Based Weighted K-Nearest Neighbor: Rather then simply taking an average of the coefficient value of k/m nearest neighbor for each of the class, if we use some heuristic function to calculate the average (like based on the distance average), we may perform better for a variety of datasets.

Related Publications

1. Harshit Dubey and Vikram Pudi,“Class Based Weighted K-Nearest Neighbor Over Imbalance Dataset”, In Proceedings of 17^{th} Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2013).

2. Harshit Dubey and Vikram Pudi, “CLUEKR : CLUstering based Efficient kNN Regression”, In Proceedings of 17^{th} Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2013).

3. Saket Bharambe, Harshit Dubey and Vikram Pudi, “BINER : BINary search based Efficient Regression”, In Proceedings of 8^{th} International Conference on Machine Learning and Data Mining (MLDM 2012).

4. Harshit Dubey, Saket Bharambe and Vikram Pudi, “BINGR : BINary search based Gaussian Regression”, In Proceedings of 4^{th} International Conference on Knowledge Discovery in Information Retrieval (KDIR 2012).

Bibliography

[1] D. N. A. Asuncion. UCI machine learning repository, 2007.

[2] H. Barreto. An introduction to least median of squares. In *Chapter contribution to Barreto and Howland,Econometrics via Monte Carlo Simulation*.

[3] S. Berchtold, B. Ertl, D. A. Keim, H.-P. Kriegel, and T. Seidl. Fast nearest neighbor search in high-dimensional space. In *Proceedings of the Fourteenth International Conference on Data Engineering*, ICDE '98, pages 209–218, Washington, DC, USA, 1998. IEEE Computer Society.

[4] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[5] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. Chapman & Hall/CRC, 1984.

[6] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap. Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In *Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD '09, pages 475–482, Berlin, Heidelberg, 2009. Springer-Verlag.

[7] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer. Smote. *Journal of Artificial Intelligence Research*, 16(1):321–357, 2002.

[8] W. Chu and S. S. Keerthi. New approaches to support vector ordinal regression. In *In ICML 05: Proceedings of the 22nd international conference on Machine Learning*, pages 145–152, 2005.

[9] D. A. Cieslak and N. V. Chawla. Learning decision trees for unbalanced data. In *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I*, ECML PKDD '08, pages 241–256, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. 10.1007/BF00994018.

[11] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, New York, 2. edition, 2001.

[12] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, New York, 2. edition, 2001.

[13] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in knowledge discovery and data mining*. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[14] K. Feng, J. Gao, K. Feng, L. Liu, and Y. Li. Active and passive nearest neighbor algorithm: A newly-developed supervised classifier. In D.-S. Huang, Y. Gan, P. Gupta, and M. Gromiha, editors, *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, volume 6839 of *Lecture Notes in Computer Science*, pages 189–196. Springer Berlin Heidelberg, 2012.

[15] S. Garcia and F. Herrera. Evolutionary undersampling for classification with imbalanced datasets: proposals and taxonomy. *Evolutionary Computation*, 17(3):275–306, 2009.

[16] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11:10–18, November 2009.

[17] B. Hamers, J. A. K. Suykens, and B. D. Moor. *Compactly Supported RBF Kernels for Sparsifying the Gram Matrix in LS-SVM Regression Models*. 2002.

[18] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*, volume 54. Morgan Kaufmann, 2006.

[19] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, 1999.

[20] D. Hosmer and S. Lemeshow. *Applied Logistic Regression*. Wiley Series in Probability and Statistics: Texts and References Section. John Wiley & Sons, 2000.

[21] J. P. Hwang, S. Park, and E. Kim. A new weighted approach to imbalanced data classification problem via support vector machine with quadratic cost function. *Expert Syst. Appl.*, 38:8580–8585, July 2011.

[22] L. Hyafil and R. L. Rivest. Constructing Optimal Binary Decision Trees is NP-Complete. *Information Processing Letters*, 5:15–17, 1976.

[23] M. Z. Jahromi, E. Parvinnia, and R. John. A method of learning weighted similarity function to improve the performance of nearest neighbor. *Inf. Sci.*, 179:2964–2973, August 2009.

[24] T. M. Khoshgoftaar, S. Zhong, and V. Joshi. Enhancing software quality estimation using ensemble-classifier based noise filtering. *Intell. Data Anal.*, 9(1):3–27, Jan. 2005.

[25] E. Kriminger, J. Principe, and C. Lakshminarayan. Nearest neighbor distributions for imbalanced classification. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1 –5, june 2012.

[26] Y. Li and X. Zhang. Improving *k* nearest neighbor with exemplar generalization for imbalanced classification. In *PAKDD (2)*, pages 321–332, 2011.

[27] O. C. Lingj and K. Liestøl. Generalized projection pursuit regression. *SIAM J. Sci. Comput.*, 20(3):844–857, Jan. 1999.

[28] R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. Number Second Edition. Wiley-Interscience, 2002.

[29] W. Liu and S. Chawla. Class confidence weighted *k*nn algorithms for imbalanced data sets. In *PAKDD (2)*, pages 345–356, 2011.

[30] C. D. Liu W., Chawla S. and N. Chawla. A robust decision tree algorithms for imbalanced data sets. In *Proceedings of the Tenth SIAM International Conference on Data Mining*, pages 766–777, 2010.

[31] G. Loizou and S. J. Maybank. The nearest neighbor and the bayes error rates. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-9(2):254 –262, march 1987.

[32] V. E. McZgee and W. T. Carleton. Piecewise regression. *Journal of the American Statistical Association*, 65(331):1109–1124, 1970.

[33] D. Montgomery, E. Peck, and G. Vining. *Introduction to linear regression analysis*. Wiley series in probability and statistics: Texts, references, and pocketbooks section. Wiley, 2001.

[34] M. Nikulin. Hellinger distance. *Encyclopedia of Mathematics.*, 2001.

[35] Y. Q. and W. X. 10 challenging problems in data mining research. *International Journal of Information Technology and Decision Making*, 5(4):597–604, 2006.

[36] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[37] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, SIGMOD '95, pages 71–79, New York, NY, USA, 1995. ACM.

[38] S. Shevade, S. Keerthi, C. Bhattacharyya, and K. Murthy. Improvements to smo algorithm for svm regression. Technical report, National University of Singapore, Control Division Dept of Mechanical and Production Engineering, National University of Singapore, 1999. Technical Report CD-99-16.

[39] H. Singh, A. Desai, and V. Pudi. Pager: parameterless, accurate, generic, efficient knn-based regression. In *Proceedings of the 21st international conference on Database and expert systems applications: Part II*, DEXA'10, pages 168–176, Berlin, Heidelberg, 2010. Springer-Verlag.

[40] A. J. Smola and B. Schoelkopf. A tutorial on support vector regression. In *NeuroCOLT2 Technical Report Series*. 1998. NC2-TR-1998-030.

[41] Y. Song, J. Huang, D. Zhou, H. Zha, and C. L. Giles. Iknn: Informative k-nearest neighbor pattern classification. In *PKDD*, pages 248–264, 2007.

[42] Y. Tang, Y.-Q. Zhang, N. V. Chawla, and S. Krasser. Svms modeling for highly imbalanced classification. *Trans. Sys. Man Cyber. Part B*, 39:281–288, February 2009.

[43] J. D. Van Hulse, T. M. Khoshgoftaar, and H. Huang. The pairwise attribute noise detection algorithm. *Knowl. Inf. Syst.*, 11(2):171–190, Feb. 2007.

[44] Y. Wang. *A new approach to fitting linear models in high dimensional spaces*. PhD thesis, Department of Computer Science, University of Waikato, Hamilton, New Zealand, 2000.

[45] Y. Wang and I. H. Witten. Modeling for optimal probability prediction. In *ICML*, pages 650–657, 2002.

[46] G. M. Weiss. Mining with rarity: a unifying framework. *Sigkdd Explorations*, 6(1):7–19, 2004.

[47] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14:1–37, December 2007.

[48] T. Yang, L. Cao, and C. Zhang. A novel prototype reduction method for the k-nearest neighbor algorithm with k $\geq$ 1. In *Advances in Knowledge Discovery and Data Mining*, volume 6119 of *Lecture Notes in Computer Science*, pages 89–100. Springer Berlin / Heidelberg, 2010.

[49] Q. Yu, Y. Miche, A. Sorjamaa, A. Guillen, A. Lendasse, and E. Séverin. Op-knn: method and applications. *Adv. Artif. Neu. Sys.*, 2010:1:1–1:6, Jan. 2010.

[50] X. Zhu and X. Wu. Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22:177–210, 2004.