# *Estimation of Error Bound for $k$-Nearest Neighbor Classifier on Multi Class Data Sets*

*Kushal Bose*

# Estimation of Error Bound for $k$-Nearest Neighbor Classifier on Multi Class Data Sets.

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

**Kushal Bose**
[ Roll No: CS-1714 ]

under the guidance of

**Dr. Swagatam Das**
Associate Professor
Electronics and Communication Sciences Unit

**Indian Statistical Institute**
**Kolkata-700108, India**

**July 2019**

*To my parents and my guide*

*"Imagination is more important than Knowledge"*

*- Albert Einstein*

## CERTIFICATE

This is to certify that the dissertation entitled **"Estimation of Error Bound for $k$-Nearest Neighbor Classifier on Multi Class Data Sets"** submitted by **Kushal Bose** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

**Dr. Swagatam Das**
Associate Professor,
Electronics and Communication Sciences Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

# Acknowledgments

I would like to show my highest gratitude to my advisor, *Associate Prof. Dr. Swagatam Das*, Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata, for his guidance and continuous support and encouragement. He has literally taught me how to do good research, and motivated me with great insights and innovative ideas.

My deepest thanks to all the teachers of Indian Statistical Institute, for their valuable suggestions and discussions which added an important dimension to my research work.

Finally, I am very much thankful to my parents and family for their everlasting supports.

Last but not the least, I would like to thank all of my friends for their help and support. I thank all those, whom I have missed out from the above list.

**Kushal Bose**
Indian Statistical Institute
Kolkata - 700108 , India.

# Abstract

A motivational problem that arises in machine learning is to estimate out-of-sample error rate of a $k$-nearest neighbor classifier. Without having any prior knowledge of distribution or any assumption of distribution it is required to estimate the maximum probability of misclassification of an unlabeled sample. Previous works include the assumption on data distribution as identical and independent distribution (i.i.d.). This method works for binary classification only. Our proposed algorithm is applicable for any data sets without having any knowledge of the underlying data distribution. Our algorithm will search the misclassification region in the data set and calculate the bound for an unlabeled test sample. Our method will always detect the class overlapping region within the data set irrespective of balanced and imbalanced. Our method is also designed for both two-class and multi-class data sets. Our experiments includes the bound validation for different scenarios. We have tested for random $k$ value and fixed range of $k$. Also verified for balanced and imbalanced data sets. We also demonstrated to find optimal sets of $k$ values where classifier error will be minimized.

**Keywords**: *Error Bound, Penalty Function, Penalty Matrix, Distance Weighted Score (DWS), Additive Score, Misclassification Region.*

# Contents

# List of Figures

# List of Tables

Table 1: List of Notations Used in Thesis

| | |
|---|---|
| $D$ | A Multi-Class Data Set |
| $n$ | Number of Samples in data set |
| $R$ | Number of Class Labels |
| $k$ | Number of Neighbors |
| $d$ | Dimension of Each Sample |
| $n_q$ | Number of Samples in $C_q$ Label |
| $frac$ | Fraction of Samples |
| $V$ | Validation Set |
| $T$ | Target Set |
| $N$ | Neighbor Set |
| $Penalty(.)$ | Penalty Function |
| $label(.)$ | Label of a Sample |
| $M$ | Misclassification Region |
| $P_{n \times R}$ | Penalty Matrix |
| $Z$ | Unlabeled Test Sample |
| $prob(i)$ | Misclassification probability of $i^{th}$ Sample |
| $Th\_Bound$ | Theoretical Error Bound |
| $Exp\_Bound$ | Experimental Error Bound |
| $EB_i$ | Theoretical Error Bound in $i^{th}$ Iteration |

# Chapter 1

# Introduction

## 1.1 Problem Statement

We have a data set $D$ and we want to apply $k$-NN classifier to determine label of test sample using training samples of $D$. Our objective is to find error bound of this $k$-NN classifier. Simply, we want to estimate the **maximum probability of misclassification** of an unlabeled test sample.

During validation phase some training examples are removed which is known as Validation Set $V$ and this samples are tested on the rest of the training samples. Therefore, selection of $V$ is done randomly and uniformly. So, it is not guaranteed that the $V$ will be selected from most overlapping region. For that reason error rate in validation set may not ensure the error bound. Therefore, we desire to estimate error bound in an alternative way.

## 1.2 Brief Survey on $k$-NN Classifier

In machine learning, a set of labeled training examples is used to develop a classifier. Each example consists of an input and a class label. The primary objective is to learn the features using the labels associated with it. This type of learning is called Supervised Learning. $k$-Nearest Neighbor algorithm is an example of supervised learning.

$k$-nearest neighbor is used to classify labeled examples. $k$NN algorithm works based on the majority voting of neighbor examples. It does not require any training phase [1]. Therefore, it is a lazy learner. It does not require any assumption of data distribution. So, it is a non-parametric classifier. It is also an example of instance based learning. For any unlabeled sample it calculates neighbor set and then determine the label.

In $k$NN classifier the only parameter is $k$ where $k$ is the number of nearest neighbors. Selection of $k$ value is very important while applying $k$-NN classifier. Randomly selection of $k$ either can give good accuracy or it may lead to bad performance. Guo *et al.* [2] showed the dependency of performance on $k$. They also showed that value of $k$ is automatically determined which is varied for different data, and is optimal in terms of classification accuracy. The construction of the method reduces the dependency on k and makes classification faster. Bhattacharya *et al.* [3] showed value of $k$ can be large as $\sqrt{n}$ where $n$ is the number of samples in data set.

Use of $k$NN is limited due to high storage of all training samples and requirement of intensive computation. Behaviour of $k$NN rule under some situation described by Sanchez *et al.* [4]. They described the situations like class overlapping, feature space dimensionality and class density. Hinneberg et al [5] discussed about the effect of high dimensionality. They suggested to select only important features and reject others as well. This will increase performance and reduces time complexity.

If data set contains categorical features then $k$-NN is not applicable. If the feature values are categorical then evaluation of metrics are not possible. It only works if all feature values are either integer or real numbers or both.

One important thing is that $k$NN classifier has dependency on the relative positions of points. This classifier works taking majority voting of nearest neighbors. The different arrangement of data points may change the neighbors position. The decision of majority voting also changes. This can affect the performance. Either the performance will be better or worse.

$k$NN algorithm can helps to extract information from imbalanced data distribution. Ahang *et al.* [6] showed the effect of under sampling on the $k$NN approach and stated different methods of choosing negative training examples.

Moreover when a data set and there is no prior information about the distribution of the data then $k$ nearest neighbor classifier comes to rescue. This classifier does not require or assumption of any distribution of the data points. it just considers its nearest neighbors and majority voting.

## 1.3 Thesis Overview

The rest of the thesis is organized as follows -
**Chapter 2**: We briefly discuss about preliminaries and $k$-nearest neighbor algorithm.
**Chapter 3**: We discuss related work, motivation and our contribution to solve challenges.
**Chapter 4**: We discuss about brief overview of proposed error bound estimation algorithms and detailed analysis related to every step used in the algorithms.
**Chapter 5**: Error Bound estimation algorithms for Two -Class and Multi-Class data sets are discussed along with pseudo codes.
**Chapter 6**: We exhibit different types of experiments on various real life data sets (small and large) and how to find optimal set of $k$ values of a $k$NN classifier.
**Chapter 7**: We summarize our work and discuss about future scope to improve the work.

# Chapter 2

# Preliminaries

## 2.1 $k$-Nearest Neighbor Algorithm

$k$-nearest neighbor algorithm is a non-parametric lazy learning algorithm [7]. The label of an unlabeled sample is determined by majority voting of the $k$ nearest neighbors. $k$-NN is mostly applied when there is little or no prior knowledge about the distribution of data.

Suppose $t$ is an unlabeled sample whose class label is to be determined. First calculate $k$ nearest neighbors using Euclidean distance metric. Consider the neighbor set is $N = \{n_1, n_2, ..., n_k\}$. Now count the number of samples in each label we get as $m_i, \forall i = 1, 2, .., R$ where $m_i$ is the count of number of samples of $i^{th}$ class from the neighbor set. Find the class label whose count is maximum. Assign that class label to that unlabeled sample t. Break ties randomly. In this way $k$-NN algorithm works.

The distance which is measured between samples can be of different metric. May be it is Euclidean, Manhattan or it is Minkowski. By using different metric results may vary [8]. In whole thesis we will use Euclidean Distance as our distance metric.

Consider the following two class example - The green colored sample need to classify (see Figure 2.1). For $k = 1$, the nearest neighbor has class label of class 1(green color). So, the unlabeled sample will assign to class 1. For $k = 3$, the nearest neighbors are two samples from class 2(red color) and one sample from class 1. Therefore, class label will be class 2.

Applications of $k$-NN can be handwritten character recognition, image recognition, video recognition. $k$-NN can also classify a voter either "will vote" or "will not vote".

## 2.2 Pros and Cons of $k$-NN Classifier

### 2.2.1 Pros

- Simple Algorithm

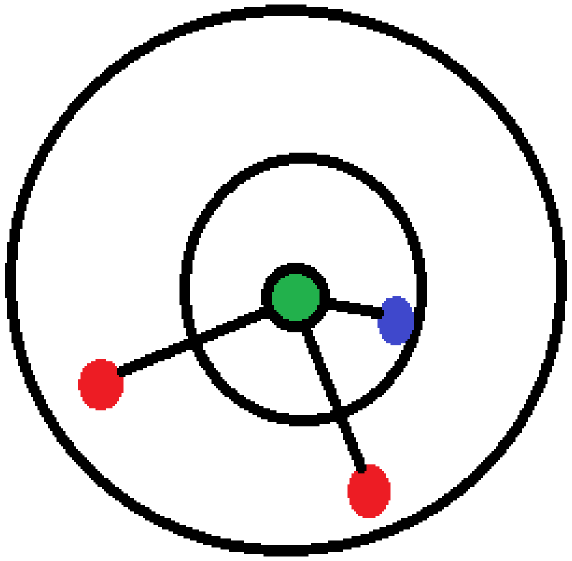


Figure 2.1: Calculating neighbor set for two class problem

- No assumption about data
- Versatile - Useful for Classification and Regression

### 2.2.2 Cons

- High Memory Requirement - Stores all training data
- For high dimensional data $k$-NN algorithm takes long time
- Sensitive to irrelevant features and the scale of the data

## 2.3 Validation Phase of $k$-NN Classifier

As discussed in the introduction of previous chapter $k$-nearest neighbor requires no training phase. It has only validation phase and test phase. First from the set of samples $D$ select randomly $n \times frac$ number of samples and store them in validation set $V$. Generally $frac$ is $10\%, 15\%, 20\%$ of all samples. For each $v \in V$ find neighbors from $D \setminus v$ and apply majority voting. Then assign the label. If predicted label matched with actual label then it is properly classified if not then it is misclassified. Store the samples which are classified in the set $V'$. Do this for all samples in validation set. Then accuracy will be $\frac{V'}{V}$.

**Remark 1** *Observe that accuracy will change every time because the set $V$ is selected randomly from the data set. So, accuracy increases or decreases abruptly. So, we need to an error bound to confirm the error rate.*

## 2.4 Two Different Types of $k$-NN Classifier

### 2.4.1 Distance Weighted k-NN (DWkNN)

In normal KNN the the final label is determined by counting the number of samples from each class and then find the label having maximum count. But in DWKNN [9] an additional weight is assigned that is inverse distance between unlabeled sample and its nearest neighbors. While counting of number samples for a particular class just add the inverse distances i.e. $\sum_i \frac{1}{w_i}$. Repeat this for all class labels. Then assign final label which class label has maximum value.

### 2.4.2 Fuzzy k-NN (FkNN)

In FKNN [10] every unlabeled sample will belong to every class label to some fractional value that is Fuzzy value. In general k-NN algorithm every sample belongs to exactly one class. But in this case every unlabeled sample is assigned a fractional value $f_i$ for $i^{th}$ class. So, $\sum_i f_i = 1$. The value $f_i$ depends on the position of samples from that class label.

## 2.5 Optimal Data Structures for $k$-NN Algorithm

k-NN algorithm takes time to find $k$ nearest neighbors. First algorithm calculates all distances between test sample and all other neighbors. This takes $O(n)$ time. To optimize this there are many data structures are there -

### 2.5.1 Kd Tree

This method uses tree like structure. It divides the higher dimensional feature space into equal halves recursively w.r.t each dimension once. While calculating test sample compares distance between two halves and which one is larger this half is fully rejected [11]. In this way searching reduces to $O(\log n)$.

### 2.5.2 Ball Tree

This also divides like kd tree but in spherical structures. The higher dimensional space is recursively divided into equal halves w.r.t each dimension once. While searching the one sphere is fully rejected [12]. It also takes $O(\log n)$ time.

# Chapter 3

# Related Work and Our Contribution

## 3.1 Related Work

There are many approaches are taken to estimate error bounds for $k$-nearest neighbor classifier. Bax [13] evaluated Probably Approximately Correct or PAC Bound for out of sample error rate of k-NN classifier. Bax showed in his paper that the PAC bound works for samples which are drawn from known identical and independent distribution (i.i.d.). Also it assumes of binary classification problem. One class is positive class and another class is negative. This method holds some training samples as validation set. This classifier is called $holdout\ classifier$. If all training samples are used then it is termed as $full\ classifier$. First bound the error rate of the $holdout\ classifier$. Now evaluate the bounding difference between *holdout* and $full\ classifier$. Combining this two bounds is the final error bound for $Full\ Classifier$. Experiments are done with 10,000 samples randomly drawn from $[0,1]^3$. Class label is determined by octant centered at $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$. Author showed the bound for different $k$ values. The bound is getting looser with increasing with the value of $k$. Finally the expected error bound is in the range of $O((k/n)^{\frac{2}{3}})$

Yunwen Lei *et al.* [14] showed the data dependent generalization error bound which has low dependency on number of classes. They developed new structural results for multi class Gaussian complexities and $l_\infty$ covering numbers which exploit the Lipschitz continuity of the loss function with respect to the $l_2$ and $l_\infty$ norm, respectively. They establish data-dependent error bounds in terms of complexities of a linear function class defined on a finite set induced by training examples, for which they show tight lower and upper bounds.

Another significant work by Devroe *et al.* [15] showed that $k$-NN produces an exponential error bound with range $O((\frac{k}{n})^{\frac{1}{3}})$.

## 3.2 Motivation

In the review while evaluating error bound there are many constraints such as assumption of distribution, number of classes should be two etc. There also exists dependency on number of class labels in the error bound. Real life data sets may not guarantee that it should follow some particular distribution and it can be of multi classes. Also data set can be imbalance. Therefore, this methods work in very limited cases. Therefore, we took this as a challenge and proposed our error bound estimation algorithms. Our algorithms will estimate the maximum probability of misclassification of an unlabeled test sample without any assumption. We developed two different methods - one for two class and another for multi classes. Apart from error bound calculation we also generate the optimal values of $k$ for which error rate will be minimized.

## 3.3 Our Contribution

Our contributions are summarized as follows -

- We have developed two different algorithms for error bound estimation - one for two-class and another for multi-class data sets.
- Using these algorithms we have detected overlapping regions lies within the data set.
- We have calculated the maximum probability of misclassification of an unlabeled test sample.
- We have done complexity analysis for both algorithms
- We performed several types of experiments on various real life data sets.
- From different error bound for different values of $k$ we assure some optimal sets of $k$ for which error rate is minimized.

# Chapter 4

# Proposed Approach for Error Bound Estimation

## 4.1 Introduction

In this chapter we will discuss every step of our proposed algorithms along with the theories. There will be theorems, lemmas etc to justify our work. This will establish the correctness of our proposed algorithms.

## 4.2 Overview of Proposed Algorithm

**Input:** A data set $D = \{x_1, x_2, \ldots, x_n\}$ of n samples with $R$ class labels. A $k$-NN classifier with predefined $k$ value.

**Output:** Return the maximum value of probability of misclassification for that $k$ value.

**Step 1:** Select some/all training samples from data set and termed them as *Target* samples.

**Step 2:** Consider $x_i$ from the set of target sample set and find its $k$ nearest neighbors from the set $D \setminus x_i$.

**Step 3:** Update penalty value for nearest neighbors for each target samples

**Step 4:** Calculate probability of misclassification for each sample using these penalty values.

**Step 5:** Sort this samples according to misclassification probability values.

**Step 6:** For class label $C_i$ evaluate weighted mean probability of top $k$ samples. Repeat this for all class and consider the maximum of them as final error bound.

## 4.3 Detailed Analysis of Proposed Approach

### 4.3.1 Target Sample Selection

Target sample means this set of samples to be selected to find their nearest neighbors from the data set. Consider collection target samples as set $T$. For small data set means number of samples may be less than 1000, then $T = n$. For large data set where number of samples is more than 5000, then $T = n * frac$. Here $frac$ is user given parameter. Value of frac is fraction of samples to be selected uniformly and randomly. So, frac can be $0.60, 0.70$ etc. Therefore $|T| \leq n$.

### 4.3.2 Penalty Function

Literally penalty means imposing some fine for doing wrong deeds. Here wrong deed means 'Misclassification'. Also 'Classification' is wrong deed but lesser penalty from misclassification. Penalty functions produces penalty values which are assigned to the neighbors of a target sample irrespective of classification / misclassification. Penalty function consists of two different scores -

(1) Distance Weighted Score (DWS)

(2) Class Wise Sample Count in Neighbor Set (*Label_Count*)

#### Distance Weighted Score

This score is assigned according to the distance of the neighbor from the target sample. After calculating $k$ nearest neighbors sort them w.r.t increasing order of distance from target sample. Then $DWS$ will be following -

$$DWS(x) = k - Pos(x) \tag{4.1}$$

where $Pos(x)$ = index of sample x in corresponding neighbor set

Now consider t is a target sample (see Figure 4.1) and $k = 3$, Therefore, it has 3 nearest neighbors which are p,q,r. Sort them w.r.t. increasing order of distance then neighbor set will looks like $N = \{p, q, r\}$. Now $Pos(p) = 0$, $Pos(q) = 1$, $Pos(r) = 2$. Their $DWS$ will be following -

$DWS(p) = k$ - $Pos(p) = k$ - $0 = k$

$DWS(q) = k$ - $Pos(q) = k$ - 1

$DWS(r) = k$ - $Pos(r) = k$ - 2

It can be observed that $DWS$ is assigned such a way that more nearer to the target sample higher the score is. For a fixed $k$, the contribution towards misclassification is measured how the neighbors is nearer to the target sample. If a sample is more nearer to the t more its contribution to the misclassification.

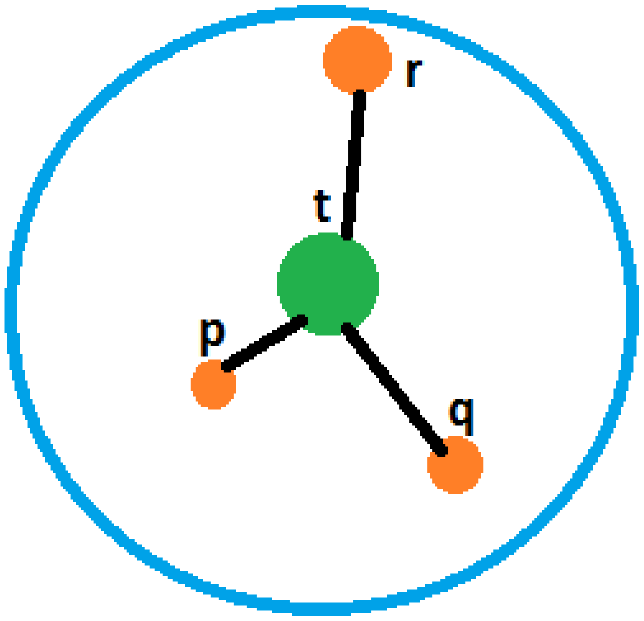


Figure 4.1: Distance weighted score for neighbors

### Class Wise Sample Count in Neighbor Set

From the neighbor set calculate the number of samples from each class label. Number of samples from $C_i$ is denoted as $Label\_Count[C_i]$. It is a part of penalty value because more the number of samples of a particular class label higher the probability that target sample will be classified to that label. So, higher the sample count of class label more contribution towards misclassification or classification. Therefore, combining this two score components the penalty function will looks like -

$$Penalty(x) = DWS(x) + Label\_Count[label(x)] \tag{4.2}$$

where $x$ neighbor of any target sample and label(.) denotes class label of the sample.

**Proposition 1** *Adding $DWS$ and $Label\_Count$ scores will lead to more accurate Penalty function*

**Proof:**

In penalty value two scores $DWS$ and $Label\_Count$ are added. The obvious question may arise about the justification of adding two terms(*Additive Scores*). Consider t is a target sample from the data set and $k = 3$ (see Figure 4.2). The neighbors are $\{p, q, r\}$. Sort them w.r.t. ascending order of distance from t we get $\{p, r, q\}$. Sample p has class label 1 where sample r and q has class label 2. So, $Pos(p) = 0$, $Pos(q) = 2$, $Pos(r) = 1$. Let target sample t has class label 1. Applying $k$-NN t will be classified as class 2. So, it is a misclassification.

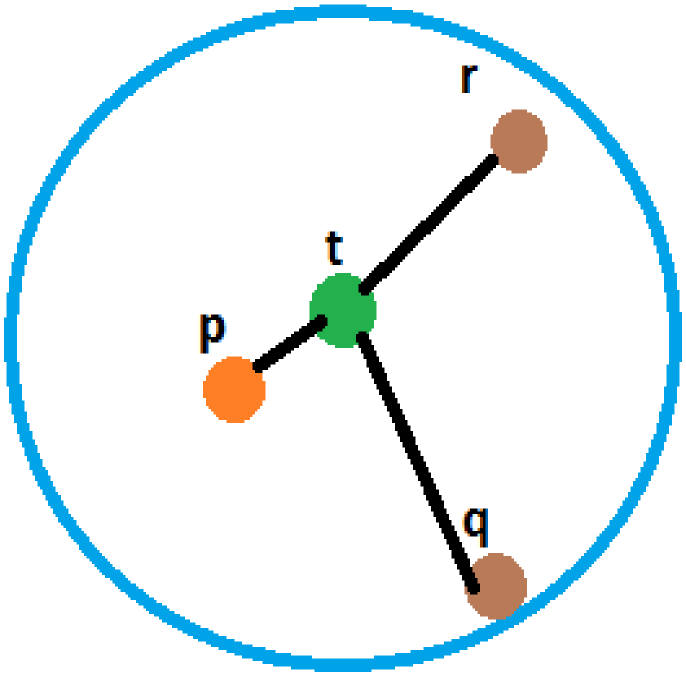


Figure 4.2: Scenario where additive scores are required

Now penalty will be assigned as misclassification occurs.

$DWS(p) = k$ - $Pos(p)$ = 3 - 0 = 3
$DWS(q) = k$ - $Pos(q)$ = 3 - 2 = 1
$DWS(r) = k$ - $Pos(r)$ = 3 - 1 = 2

The label counts are like below -

$Label\_Count[C_1] = 2$
$Label\_Count[C_2] = 1$

Samples q and r are highly responsible for the misclassification of target sample t. So, eventually they should get higher penalty value. But $DWS$ are lesser than expected for $q$ and $r$ but adding $DWS$ and $Label\_Count$ both will enhance the penalty values to the expectation. So, adding these two score we get final penalty value -

$Penalty(p) = DWS(p) + Label\_Count[l(p)] = 3 + 1 = 4$
$Penalty(q) = DWS(q) + Label\_Count[l(q)] = 1 + 2 = 3$
$Penalty(r) = DWS(r) + Label\_Count[l(r)] = 2 + 2 = 4$

Observation is that the meaningful change in penalty values after using additive score technique. Sample q is farthest from all others and initially it has low penalty value but it should have higher penalty value as it contributes more to misclassification. Using additive score this will give 3 points where sample p is nearest has 4 penalty points i.e. difference is 1. But initially this difference is 2 but now difference is 1. That is the improvement has occurred. For sample $r$ initially penalty value is 2 which is lesser than penalty value of sample $p$. But $r$ should have higher value than $p$. After using additive scores penalty value of $r$ becomes 4 which is same as penalty value of $p$. So, it is justifiable why to use additive scores. □

### 4.3.3 Penalty Matrix Generation

Penalty Matrix is a matrix which will store the penalty value of the nearest neighbors of a target sample. Penalty matrix is $P$ has dimension of $n \times R$ where n = no of samples, R = number of classes.

Initially all values of $P$ will be zero.

$$P_{n\times R} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \cdots & & & \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

Select any target sample $x_i$ from $T$. Find its $k$ nearest neighbors from data set $D\backslash x_i$. Let its neighbor set is $N = \{n_1, n_2, ..., n_k\}$. If $x_i$ misclassified then its neighbors are going to get some penalty value. If $x_i$ got classified then its neighbor samples will get less penalty values. This values will be stored in penalty matrix.

Suppose we will find $k$ nearest neighbors of $x_i$. Assume it got misclassified and $N$ will be the set of neighbors. So, penalty update will be like this -

```
Let $N = \{n_1, n_2, ..., n_K\}$
for each $h \in N$
        if label(h) = j then
                Update $P[i][j]$ += $Penalty(h)$
```

Where $P[i][j]$ is the penalty value of a neighbor whose class label is j of $i^{th}$ target sample.

### 4.3.4 Misclassification Probability for Each Sample

Using penalty matrix we can calculate the probability of misclassification of each sample lies in the data set. Two different ways are proposed for two class and multi class problem separately. This will be described in details in next chapter (Chapter 4).

### 4.3.5 Sorting of Misclassification probability

From previous step, we have misclassification probability value for each sample. Sort them in ascending order. Higher the probability means that sample is highly probable to misclassify a sample.

### 4.3.6 Error Bound Calculation

Before calculating the error bound let us define the following term -

**Definition 1** *Misclassification Region*

Assume a data set as a collection of points in higher dimensional space (i.e. feature space). The region where two or more class labels overlaps is termed as Misclassification Region. If any test sample appears in this region then chance of misclassification increases as there are lots of samples from different class labels will become member of neighbor set. The dominance of other class labels will increase which lead to misclassification.

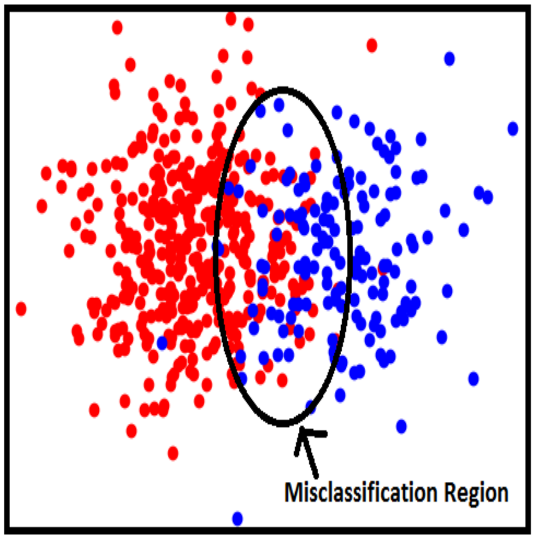


Figure 4.3: Overlapping Region

Consider an example (see Figure 4.3) where two class labels exists (red and blue label). The elliptical curve is bounding the class overlapping region. In every sample in this region will be mostly misclassified as label dominance will increase. For simplicity for every sample we select from misclassification region, its neighbor set will contain samples from both class label almost equally. So, probability of misclassification of those selected sample will increase. But if we move towards left the neighbor set will contains more more red labeled sample thus misclassification probability reduces. Same argument is applicable for moving towards right.

Our proposed method will find this misclassification regions and identify samples whose misclassification probability is high. For multi class data set there can be more complex misclassification regions. But this method will always detect this regions accurately.

**Theorem 1** *Samples having higher misclassification probabilities tends to be closer than samples having lower probabilities.*

**Proof:**
Suppose M is a region within the data set where samples are of high probabilities. Probability is high means the sample participates in more misclassifications than a sample whose probability is lower than it. So, this region is a collection of samples which participates more misclassification than classification. Therefore, this region is $Misclassification\ Region$.

Assume M contains some set of samples $M' \subset M$ whose probability is very lower than others samples in $M \setminus M'$. Samples in $M'$ participates in some classifications. So, there are classifications occurred with some samples. But it is a misclassification region. So, it is a contradiction, Thus high probability valued samples will be cluster together. If we move away from misclassification region the value of the probability will decrease and probability of classification will increase. □

**Theorem 2** *For any sample x in data set $Penalty(x) \in [2, 2k]$*

**Proof:**
Penalty function is defined as

$$Penalty(x) = DWS(x) + Label\_Count(label(x))$$

The $DWS(x)$ score will vary from 1 to $k$. It implies

$$1 \leq DWS(x) \leq k \tag{4.3}$$

The $Label\_Count[label(x)]$ score can have 0 to $k$ number of samples. Therefore,

$$0 \leq Label\_Count(label(x)) \leq k \tag{4.4}$$

Adding inequalities 4.3 and 4.4 we get

$$DWS(x) + Label\_Count(label(x)) \leq 2k$$

$$Penalty(x) \leq 2k \tag{4.5}$$

The upper bound is proved. Now its time to show the lower bound. Its simply can be 1 but we will show that penalty value can never be 1. If any sample $x$ is member of neighbor set then its $Label\_Count(x)$ will be 1. Also minimum value of $DWS(x)$

is 1 when $Pos(x)$ will be $(k-1)$ that is sample $x$ is the farthest neighbor in neighbor set. So, minimum value of $Penalty(x)$ will be 2.

$$Penalty(x) \geq 2 \tag{4.6}$$

Hence both side is proved. □

In last step we are going to evaluate error bound. Assume Z is an unlabeled test sample whose label will be determined using the given training samples by applying $k$NN algorithm. We are interested to find the maximum probability of misclassification of sample Z. Using Theorem 1 top $k$ probabilities will be nearer to each other thus $Z$ have a chance to get them as its nearest neighbors. The probability will be maximum if **Z has neighbor set containing the samples whose probabilities are top most** $k$.

Assume Z has neighbor set $N = \{n_1, n_2, ..., n_k\}$. Each $n_i$ will be from top $k$ probability. We already have the sorted list of probabilities. There are $R$ class labels are there. So, actual label can be of Z is $C_i$ but it can be misclassified as $C_j$ $\forall j \neq i$.

For class label $C_1$ consider top $k$ probabilities. But in top $k$ samples there are many other samples whose class labels are different from $C_1$. Therefore, take a count for each label and check if number of samples from $C_1$ is at least $\lfloor \frac{K}{R} \rfloor + 1$ then calculate the error bound (for details refer Chapter 4), otherwise reject. In this way consider all possibilities where misclassified class label will be $C_1$. Then repeat this process for all other classes . Store this $R$ different error bounds for each class and return the maximum of all. The maximum value is the desired maximum probability of misclassification of $Z$.

# Chapter 5

# Error Bound Estimation : Two-Class and Multi-Class Algorithms

We have designed two different approach for two-class and multi-class data sets. Both algorithms differ in penalty update, probability calculation which will be discussed in detail in following two sections.

## 5.1 Two-Class Algorithm

When it is a case of binary classification this algorithm will help to estimate error bound when k-NN classifier is applied. Let's assume the classes are $C_1$ and $C_2$. Now below steps are shown -

**Step 1 :** Select the set of target samples. Refer 4.3.1 for details

**Step 2 :** In this step penalty values will be updated for the neighbors of the target samples. Declare the penalty matrix $P$ in the following way

$$P_{n \times 2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \cdots & \\ 0 & 0 \end{bmatrix}$$

The first column stores penalty values and second column stores the number of times the samples participates in misclassification or classification. As in two class problem one sample can either be classified or misclassified. There is no any other possibility.

We assume 'Classification' is a complement of 'Misclassification'. So, whatever the penalty will give for misclassification its complement penalty will be assigned for classification.

Previously we have defined $DWS(x) = k$ - $Pos(x)$. Now define

$$\overline{DWS}(x) = 1 + Pos(x) \tag{5.1}$$

As $Pos(x)$ is the index of x in sorted neighbor set. So, $Pos(x)$ will vary from 0 to $(k-1)$. Then $DWS(x)$ will vary from 1 to $k$. Similarly $\overline{DWS}(x)$ will vary from 1 to $k$. Add them it will be -

$$DWS(x) + \overline{DWS}(x) = k + 1 \tag{5.2}$$

which is constant. If $DWS$ increases then $\overline{DWS}$ decreases and vice-versa. It implies that misclassification penalty should always greater than classification penalty value. Therefore, $DWS$ is used in misclassification and $\overline{DWS}$ is for classification.

Another penalty component is $Label\_Count$ which is simply count of samples of a fixed class label in the neighbor set. So, it will vary 0 to $k$. Now, define complement of this will be

$$\overline{Label\_Count}(x) = k - Label\_Count(x) \tag{5.3}$$

Higher the value of $Label\_Count$ lower the value of $\overline{Label\_Count}$. Due to the same reason stated above it is justified to use $Label\_Count$ in misclassification and $\overline{Label\_Count}$ in classification.

Therefore, for each misclassification assign penalty to its each neighbors -

$$Penalty(x) = DWS(x) + Label\_Count(x) \tag{5.4}$$

For each classification assign penalty to its each neighbor

$$Penalty(x) = \overline{DWS}(x) + \overline{Label\_Count}(x) \tag{5.5}$$

Let t is a $i^{th}$ target sample. Its neighbor set is $N = \{n_1, n_2, ..., n_k\}$. Assume label of t is $C_1$. Now apply k-NN and update penalty values shown in below algorithm for both classification and misclassification

**Algorithm 1** Penalty Update for Two Class Problem

```
1: if $t.MISCLASSIFIED = True$ then
2:     for $h \in N$ do
3:         $P[i][0] \leftarrow P[i][0] + k - Pos(h) + Label\_Count(label(h))$
4:         $P[i][1] \leftarrow P[i][1] + 1$
5:     end for
6: end if
7: if $t.CLASSIFIED = True$ then
8:     for $h \in N$ do
9:         $P[i][0] \leftarrow P[i][0] + 1 + Pos(h) + k - Label\_Count(label(h))$
10:        $P[i][1] \leftarrow P[i][1] + 1$
11:    end for
12: end if
```

**Step 3 :** Using penalty values from Penalty matrix, we will calculate sample wise misclassification probability. To calculate the misclassification probability of each sample first we need to calculate the probability of a sample which its contributes in either classification or misclassification. Using Theorem 2 the maximum value of *Penalty* is $2k$.

Probability in each participation = $\frac{Penalty(x)}{2k}$

Probability of misclassification of each sample is

$$\frac{\text{Total probability of all participations}}{\text{Total number of participations}}$$

Denote prob(i) as probability of $i^{th}$ sample

$$prob(i) = \frac{P[i][0]}{P[i][1] * (2k)} \tag{5.6}$$

**Step 4 :** Sort the probabilities in ascending order along with the label associated with it. Store it in the array *sorted_prob*.

**Step 5 :** in this step we will estimate maximum probability of misclassification of the unlabeled test sample Z. Sample $Z$ can be misclassified as either class label $C_!$ or $C_2$. Assume Z will be misclassified as $C_2$. So, consider top $k$ probabilities from *sorted_prob* array with the condition that in top $k$ values the number of samples of class label $C_2$ should be at least $\lfloor \frac{k}{2} \rfloor + 1$. Either $Z$ will not get the label $C_2$. Consider all such scenarios by increasing the number of samples of label $C_2$ by one and calculate total probabilities for each such scenario. The probability is calculated as below

$$weighted\_prob(C_2) = \frac{\sum_j sorted\_prob[j](k-j)}{\sum_j (k-j)} \tag{5.7}$$

There will be $(k - \lfloor\frac{k}{2}\rfloor - 1)$ number of iterations. In each iteration store the obtained *weighted_prob* value. After loop ends report the maximum value as $bound(C_2)$. Do same for class $C_!$. Store bound as $bound(C_1)$.

Therefore, final error bound is

$$Th\_Bound = max\{bound(C_1), bound(C_2)\}$$

## 5.2 Multi-Class Algorithm

For multi-class data sets this algorithm will assist to estimate the error bound. Data have $R$ class labels as $C_1, C_2, , , , , C_R$.

**Step 1 :** Select set of target samples. For details refer 4.3.1

**Step 2 :** To update penalty matrix first we need to build a new type of penalty matrix. In new penalty matrix as usual there are n rows and R columns.

$$P_{n\times R} = \begin{bmatrix} [0,0] & [0,0] & ... & [0,0] \\ [0,0] & [0,0] & ... & [0,0] \\ \cdots & & & \\ [0,0] & [0,0] & ... & [0,0] \end{bmatrix}$$

Each P[i][j] is an ordered pairs [0,0] which are initialized by zeros.

P[i][j][0] = Penalty is added here when a sample of class label $C_j$ becomes a nearest neighbor in a misclassification of $i^{th}$ sample in data set.

P[i][j][1] = Penalty is added here when a sample of class label $C_j$ becomes a nearest neighbor in a classification of $i^{th}$ sample in data set.

$\overline{\text{The penalty}}$ function and complement scores $DWS$ , $\overline{DWS}$, $Label\_Count$, $\overline{Label\_Count}$ all will be same as discussed in section 4.1 of previous algorithm.

Let t is a $i^{th}$ target sample and apply k-NN algorithm. We got N as its neighbor set $N = \{n_1, n_2, ..., n_k\}$. There can be two possibilities either t will be misclassified or classified. Lets see how to update penalty for both cases.

**Algorithm 2** Penalty Update for Multi Class Problem

```
1: if $t.MISCLASSIFIED = True$ then
2:     for $h \in N$ do
3:         if $label(h) = label(t)$ then
4:             $P[i][label(t)][1] += 1 + Pos(h) + k - Label\_Count(label(h))$
5:         else
6:             $P[i][label(t)][0] += k - Pos(h) + Label\_Count(label(h))$
7:         end if
8:     end for
9: end if
10: if $t.CLASSIFIED = True$ then
11:     for $h \in N$ do
12:         if $label(h) = label(t)$ then
13:             $P[i][label(t)][1] += 1 + Pos(h) + k - Label\_Count(label(h))$
14:         else
15:             $P[i][label(t)][0] += k - Pos(h) + Label\_Count(label(h))$
16:         end if
17:     end for
18: end if
```

**Step 3 :** In this step we will calculate misclassification probability of each from penalty matrix. As all penalty values are positive if P[i][j][0] is greater than zero then at least once $x_i$ participates in misclassification. Also if P[i][j[1] greater than zero then sample $x_i$ participates in classification for at least once.

For $i^{th}$ sample $x_i$ has class label $C_p$. We will consider $i^{th}$ row of the penalty matrix P. Total penalty of the class label other than $C_p$ is sum of ordered pairs of a column which is -

$$penalty(C_q) = P[i][q][0] + P[i][q][1] \forall q \neq p \tag{5.8}$$

The probability of misclassification of a sample $C_p$ which participates in the misclassification of a target sample of class label $C_q$ is represented as -

$$prob[label(x_i) = C_p | label(t) = C_q] = \frac{P[i][q][0]}{P[i][q][0] + P[i][q][1]} \tag{5.9}$$

Simplifying it becomes -

$$prob[C_p|C_q] = \frac{P[i][q][0]}{P[i][q][0] + P[i][q][1]} \tag{5.10}$$

Total probability of sample $x_i$ is represented as prob(i). Observe that a class label $C_p$ can misclassify all (R-1) other classes. So, applying total probability theorem -

$Prob(A) = \sum_j P(A|B_j)P(B_j)$

$$prob(i) = \sum_{q \neq p} prob(C_p|C_q)P(C_q) \tag{5.11}$$

Therefore, sample wise probability of misclassification will be -

$$prob(i) = \sum_{q \neq p} \frac{P[i][q][0]}{P[i][q][0] + P[i][q][1]} (\frac{n_q}{n}) \tag{5.12}$$

where $P[i][q][0] + P[i][q][1] \neq 0$, $P(C_q)$ is the probability of a sample of $q^{th}$ class label occurring in the data set and $n_q$ is the number of samples of $q^{th}$ class label.

**Step 4 :** Sort the sample wise misclassification probability in ascending order. Store it in a array *sorted_prob* along with the corresponding class labels.

**Step 5 :** This step is involved to estimate the error bound. Consider Z as an unlabeled test sample. The actual class label can be either $C_1$ or $C_2$ ,... or $C_R$. If actual label is $C_1$ then it can misclassified as $C_2, C_3, ..., C_R$ rest (R-1) classes.

Suppose Z has actual label of $C_p$. So, it can misclassify other (R - 1) classes. We will find error bound for each R classes. Suppose we want to find error bound for class $C_1$.

- First consider top $k$ probabilities along with labels.
- Create an array *l_count* of size R to store the count of samples from each class. All values are initialized with zero.
- Initialize a variable $count = \lfloor \frac{k}{R} \rfloor + 1$ to store the count for samples of class $C_1$.
- In every iteration, increase *count* by 1 and search for top *count* number of samples of class label $C_1$ from *sorted_prob*. After searching this find other top ($k$ - *count*) samples of other class labels (not from $C_1$) again from the beginning. Here store the count of samples from other class labels in *l_count* array. Check whether maximum from *l_count* is crossing *count* value. If this happens reject this step and go to next iteration with increasing the *count* variable by 1.
- The number of iterations will be $(k - \lfloor \frac{k}{R} \rfloor - 1)$.
- The probability is calculated in each iteration below -

$$weighted\_prob(C_1) = \frac{\sum_j sorted\_prob[j](k-j)}{\sum_j (k-j)} \tag{5.13}$$

where j will run from 0 to (k - 1). Here an additional weight (k - j) is given to assure priority to top misclassification probability and also there will be dependency on $k$.

- After the loop is over report the maximum of all values of $weighted\_prob(C_1)$.
- Repeat the above steps for all other classes and store estimated bound in the array *bound* of size $R$.
- Now assign one class to $Z$ and add rest of the bounds. If $C_1$ is the actual bound then sum the other (R - 1) bound value to the variable $bound(C_1)$. In this way calculate bound for other classes. This will looks like $\{bound(C_1), bound(C_2), ..., bound(C_R)\}$.
- Normalize these bound value by dividing (R-1). it becomes $\frac{bound(C_i)}{(R-1)}$ $\forall i \in [1, R]$ . The required error bound i.e. maximum probability of misclassification of unlabeled sample $Z$ is

$$Th\_Bound = max\{\frac{bound(C_i)}{(R-1)}\}\forall i \in [1, R]$$

## 5.3 Complexity Analysis

We will analysis performance of the both algorithms -

**Two-Class Algorithm**

1. First generate target set which is $T$
2. For every $t \in T$ we need to find $k$ neighbors and then sorting of neighbor set. To find $k$ nearest neighbors first it will calculate distance with all other neighbors and then sorting will be done . Distance calculation may take $O(d)$ where $d$ is number of features. If $d \geq n$ then time complexity will be $O(nd + n \log n)$. For simplicity we consider $d << n$ then time complexity becomes $O(n + n \log n) \approx O(n \log n)$.
3. Now penalty matrix will be updated. The outer loop will continue for $|T|$ times. Within this loop the loop in step 2 will occur. Also we are checking every target sample either classified or misclassified and updating penalty values of their $k$ neighbors. So, this will take time $O(k)$. The total complexity will be $O(k|T|n \log n)$
4. Now we will calculate misclassification probability. For every sample in data set we will calculate probability, Therefore complexity will be $O(n)$
5. We will sort this probability values. This will take time of $O(n \log n)$
6. We will calculate final error bound considering top $k$ probabilities. First loop will run $k - \lfloor \frac{k}{R} \rfloor - 1$. Within this loop we will search top $k$ samples along with other class too (details in 4.1). In worst case the inner loop can go up to $O(n)$ time

7. Combining steps (4), (5), (6) the time complexity will be $O(n + n\log n + n) \approx O(n\log n)$. Combining steps (2) and (3) we get $O(n\log n + k|T|n\log n) \approx O(k|T|n\log n)$. Combining above two finally we will get $O(n\log n + k|T|n\log n) \approx O(k|T|n\log n)$. If we consider all samples initially then $|T| = n$. Complexity will be $O(n^2 k\log n)$

### Multi-Class Algorithm

The steps are discussed for two class algorithm are same for multi class algorithm. There are some difference in penalty update and data structures rest is same. Loop runs exactly same range as two class. Therefore time complexity will be the same. The penalty update and probability calculation are of $O(1)$ time i.e. constant time operation. So, final time complexity will be $O(k|T|n\log n)$. If we use all training samples then $|T| = n$ and complexity will be $O(n^2 k\log n)$.

## 5.4 Algorithms on Large Data Sets

### Problem

Section 4.1 and 4.2 have discussed about the algorithms for two and multi class data sets separately along with pseudo code. According to complexity analysis in Section 4.3 it can be observed that it depends on the number of samples in the data set. If the number of samples is above 5000 or more than it then the algorithm may takes several minutes. Sometimes long duration is not desired. So, the obvious question arises that how to resolve this issue.

### Suggested Solution

In both of the algorithms many steps are in common such as

1. Target Sample Selection
2. Update the Penalty matrix
3. Sample Wise Misclassification Probability Calculation
4. Bound Calculation

Each of these steps takes time due to large number of training samples in the data set. So, if we can select some part of the training samples randomly and uniformly from the data set and repeating this step again and again until it converges.
Let's discuss this in detail

- First take two user input of $num\_iteration$ and $frac$. The first one means how many iteration the algorithm will run. The second input is the what fraction of samples to use in the algorithm.

- In every iteration the algorithm will output a error bound as expected. Assume in $i^{th}$ iteration the error bound is $EB_i$ and in $(i+1)^{th}$ iteration the error bound is $EB_{i+1}$.
- For a pre-defined $\epsilon$ the convergence criteria will be -

$$EB_{i+1} - EB_i < \epsilon \tag{5.14}$$

- One thing to remember that satisfaction of convergence criteria is not always perfect solution because error curve is **not a monotonic curve**. It can suddenly increase or decrease. Suppose in some iterations it converges but in stead of stopping if we continue some more iterations error bound can increase. So, convergence does not guarantee perfect solution.

**Remark 2** *Using all samples in the algorithm may give you accurate result but it can take very long time which is not expected. So, taking randomly some samples and run the algorithm again and again may not yield perfect result but can give approximate result which is time saving too. The estimated bound can be accurate by changing num_iteration and $frac$ variables which is totally experimental.*

# Chapter 6

# Experimental Study of Error Bound Algorithms

To check performance of proposed algorithms we have done four types of experiments and from results we infer optimal set of $k$ values

- Bound Validation for Random $k$ Value
- Bound Validation for Imbalanced Data Sets.
- Bound Validation for Overlapping Classes
- Bound Validation for Range of $k$

## 6.1 Calculation of Experimental Error Bound

Bound validation implies that the estimated error bound will always be greater than or equal to experimental error bound. First we will discuss the method to estimate experimental error bound. For experiment with $k$-NN classifier, python based inbuilt package is used. The inbuilt model takes input size of validation set which will be taken out from data set itself by using a $train - test - split()$ function. Another input is value of $k$. Let $V$ is set of validation set and $k$-NN is applied. the error rate is calculated as

$$err_i = \frac{\text{no of classified samples}}{|V|} \tag{6.1}$$

where $err_i$ = error rate in $i^{th}$ iteration. The experimental error rate will be calculated over many iterations. Suppose number of iterations is $I$. So, average error rate will be

$$exp\_bound = \frac{\sum_{i=1}^{I} err_i}{I} \tag{6.2}$$

In experiment, $|V|$ is taken as $0.10, 0.15, 0.20$ portion of total samples. For each value of V we have considered I as 100, In this way we evaluated experimental error bound.

## 6.2 Data Set Source Details

Whatever results will be shown below are tested over 50 different real life data sets which are taken from KEEL [16], UCI [17] data set repository. We will exhibit the results for some of them (10 data sets) which are chosen randomly from all tested data sets.

## 6.3 Experimental Results

In this section we will discuss details of our performed experiments along with figures, tables and observations.

### 6.3.1 Bound Validation for Random $k$ Value

For a data set D applying proposed algorithm we got theoretical bound as $Th\_bound$ and experimental bound *Exp_bound*. In this case value of $k$ will be chosen randomly between $[1, n/2]$. For data sets details see Table 6.1

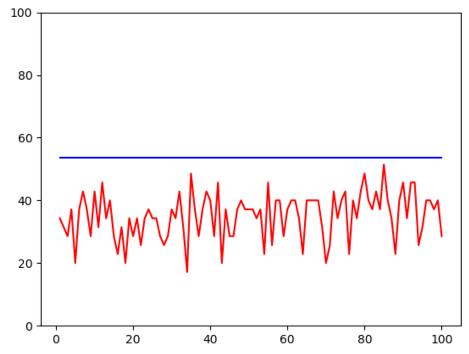
Table 6.1: KEEL Real Life Data Sets

| No | Name | Samples | Features | Classes |
|---|---|---|---|---|
| 1 | Bupa | 345 | 6 | 2 |
| 2 | Haberman | 306 | 3 | 2 |
| 3 | Heart | 270 | 13 | 2 |
| 4 | Shuttle-6 | 230 | 9 | 2 |
| 5 | Poker-9 | 244 | 10 | 2 |
| 6 | Wine | 178 | 13 | 3 |
| 7 | Glass | 214 | 9 | 7 |
| 8 | Ecoli | 336 | 7 | 8 |
| 8 | Led7digit | 500 | 7 | 10 |
| 10 | Vowel | 990 | 13 | 11 |

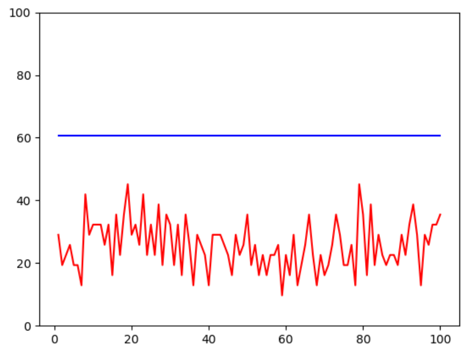
Estimated Theoretical Bound and Experimental Bound are given below. For details Refer Table 6.2

Table 6.2: Results Achieved for Random Values of $k$

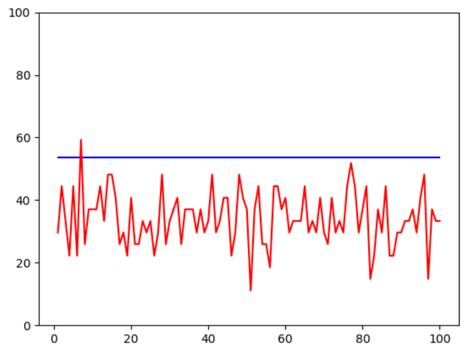
| No | Name | Th_Bound | Exp_Bound | Deviation | Value of K |
|---|---|---|---|---|---|
| 1 | Bupa | 53.04 | 35.08 | 17.96 | 87 |
| 2 | Haberman | 60.62 | 25.71 | 34.91 | 43 |
| 3 | Heart | 53.72 | 34.19 | 19.53 | 113 |
| 4 | Shuttle-6 | 36.41 | 4.04 | 32.37 | 97 |
| 5 | Poker-9 | 46.72 | 3.52 | 43.20 | 37 |
| 6 | Wine | 69.78 | 28.33 | 41.45 | 37 |
| 7 | Glass | 73.4 | 63.64 | 9.80 | 103 |
| 8 | Ecoli | 78.68 | 34.15 | 43.53 | 123 |
| 8 | Led7digit | 80.0 | 43.44 | 46.56 | 178 |
| 10 | Vowel | 90.88 | 86.42 | 4.46 | 399 |



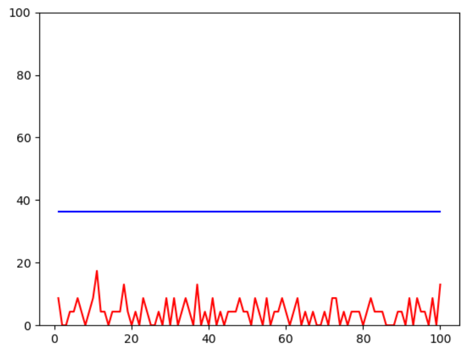
((a) ) Bupa



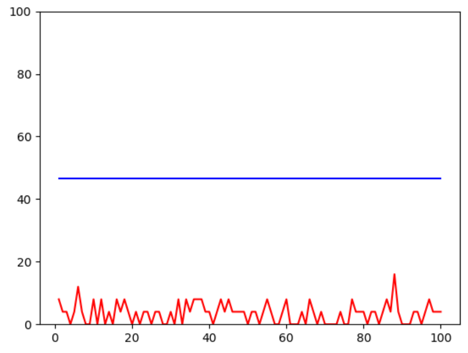
((b) ) Haberman



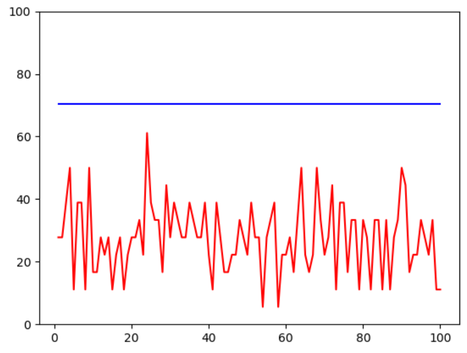
((c) ) Heart



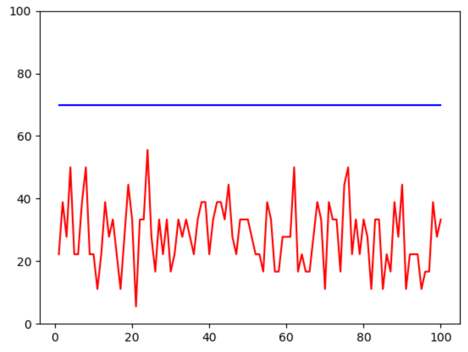
((d) ) Shuttle-6



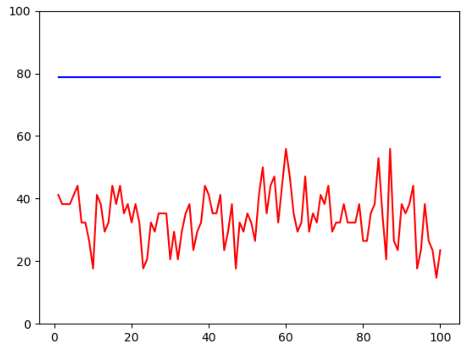
((e) ) Poker-9



((f) ) Wine



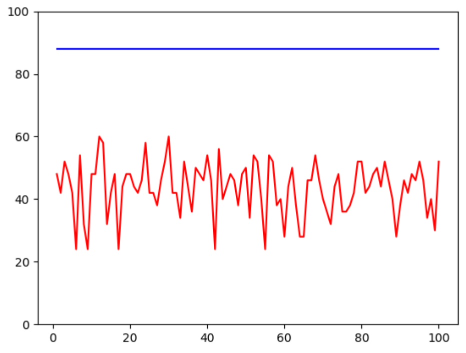
((g) ) Glass



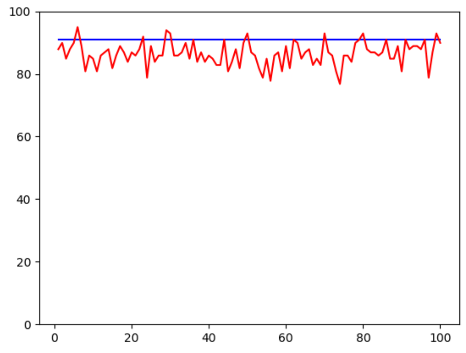
((h) ) Ecoli

## Observation

From the figures the theoretical bound is always greater than the average experimental bound. The experimental error rate is measured 100 times. Every time the bound abruptly changes which we can not predict. Actually we cant assert about what will be the error rate in the next iteration. But the theoretical bound is fixed for all iterations. Here is the success of the theoretical bound. We guarantee that how many iterations can we run but it will never cross the theoretical value. So, user will get an idea about how the $k$-NN algorithm will behave if we use that $k$ value.



((i) ) Led7digit



((j) ) Vowel

🟥 Experimental Bound

🟦 Theoretical Bound

Figure 6.1: Bound Validation for Random $k$ Values

### 6.3.2 Bound Validation for Imbalanced Data Sets

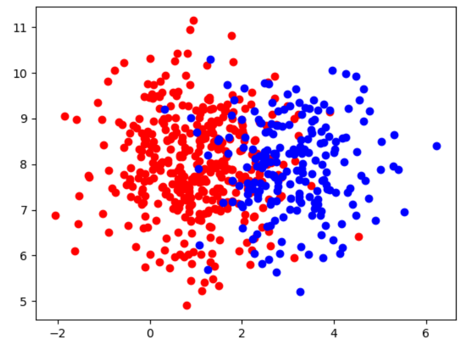
handling imbalance data sets are very challenging problem in Machine Learning or Deep learning. So, We also verified how our proposed algorithm will perform on different imbalanced data sets. We carried out the experiments by considering two classes. The samples of Class 1 is fixed and number of samples of Class 2 are varied such that imbalance ratio will vary from 2 to 80. For every case theoretical and experimental bounds are calculated. Refer Table 6.3 for randomly generated imbalance data

Table 6.3: Synthetic 2-D Imbalance Data Sets

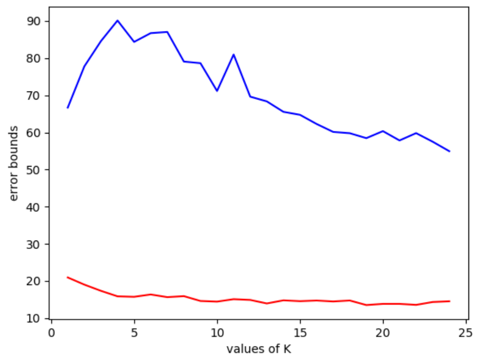
| Class1 | Class2 | Imbalance Ratio | Th_Bound | Exp_Bound | Deviation |
|---|---|---|---|---|---|
| 400 | 200 | 2 | 70 | 15 | 55 |
| 400 | 150 | 2.67 | 65 | 13 | 52 |
| 400 | 100 | 4 | 73 | 13 | 60 |
| 400 | 50 | 8 | 65 | 8 | 57 |
| 400 | 25 | 16 | 65 | 5 | 60 |
| 400 | 15 | 26.67 | 63 | 3 | 60 |
| 400 | 10 | 40 | 51 | 3 | 48 |
| 400 | 5 | 80 | 33 | 1 | 32 |

## Observation

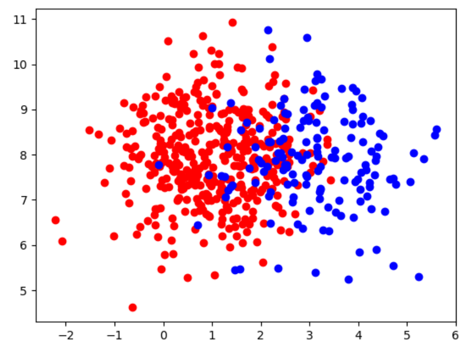
The various types of imbalance data sets are tested and results are given below. Every plot of theoretical bound and experimental bound the pattern is identical. The curve structure is independent of imbalance ratio. The imbalance ratio gradually increasing but curve patterns are still same. In the beginning bound deviation is high and when value of $k$ will increase deviation will gradually decreases. This is happening due to the detection of misclassification region. The algorithms will search the high error probability region whatever may be the imbalance ratio. For high imbalance ratio if the overlapping region is small but still proposed algorithm will identify this region properly.



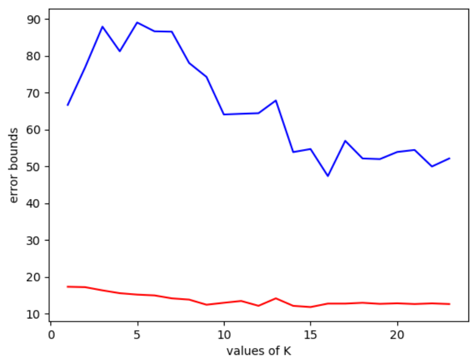
((a) ) Imbalance Data 1



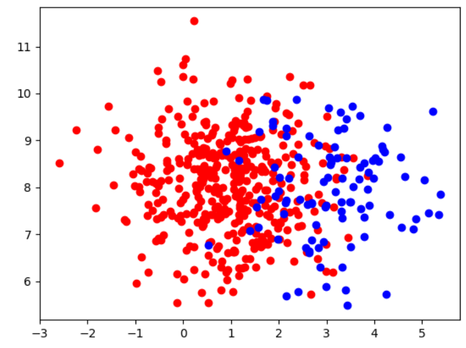
((b) ) Plot 1



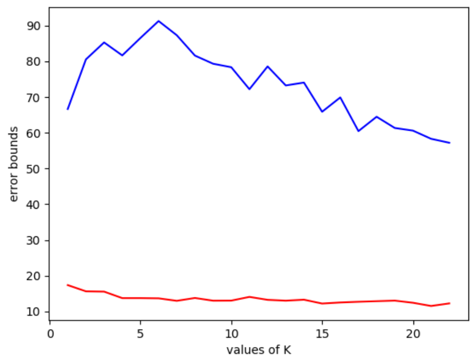
((c) ) Imbalance Data 2



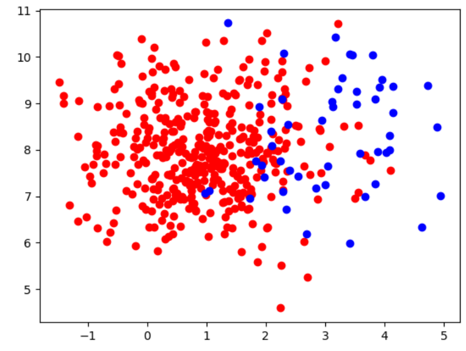
((d) ) Plot 2



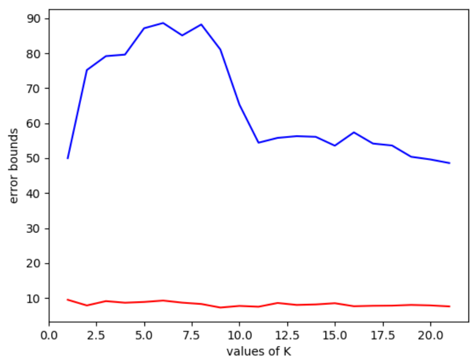
((e) ) Imbalance Data 3



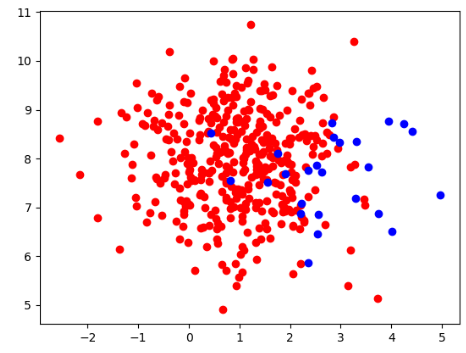
((f) ) Plot 3



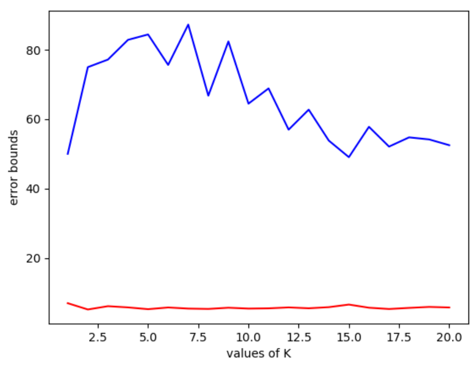
((g) ) Imbalance Data 4



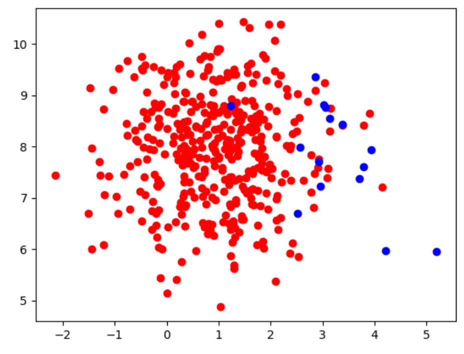
((h) ) Plot 4



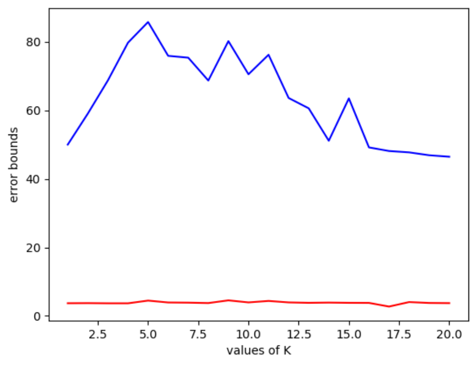
((i) ) Imbalance Data 5



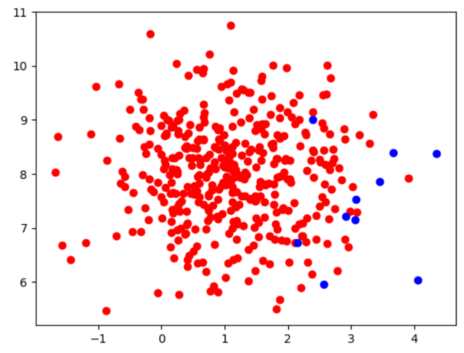
((j) ) Plot 5



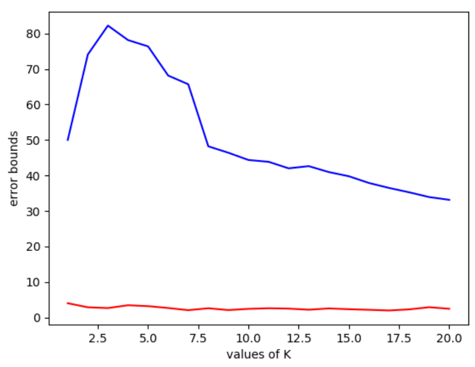
((k) ) Imbalance Data 6



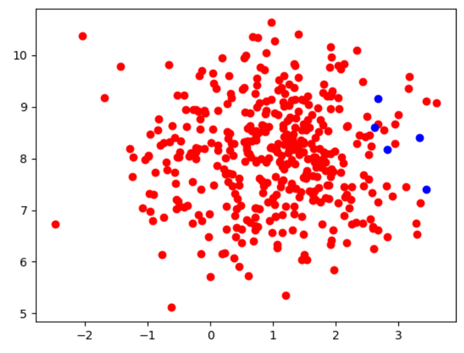
((l) ) Plot 6



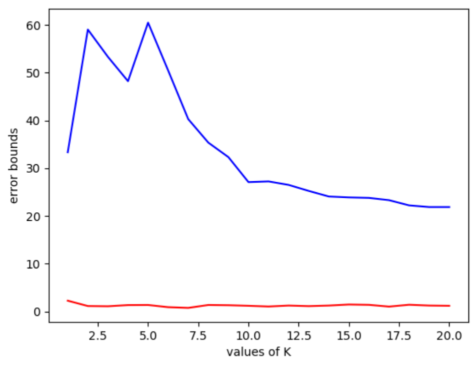
((m) ) Imbalance Data 7



((n) ) Plot 7



((o) ) Imbalance Data 8



((p) ) Plot 8

🟥 Experimental Bound
🟦 Theoretical Bound

Figure 6.2: Bound Validation for Synthetic Imbalanced Data Sets

## 6.3.3 Bound Validation for Overlapping Classes

Assume this is a two class problem. Samples of classes are in spherical shape. Suppose distance between centers of their spherical clusters is increased then how the bound deviation will vary. Class 1 has 500 samples and Class2 has 450 samples. For data set details refer Table 6.4

Table 6.4: Synthetic 2-D Class Overlapping Data

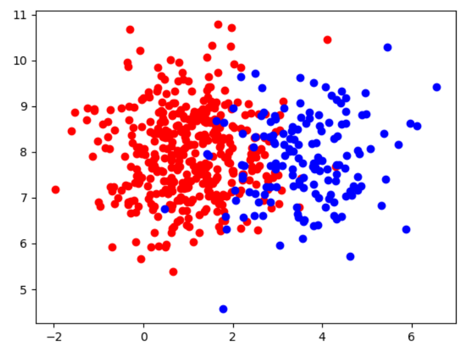
| Distance of Centers | Th_Bound | Exp_Bound | Deviation |
|---|---|---|---|
| 2 | 87 | 16 | 71 |
| 2.5 | 73 | 9 | 64 |
| 3.5 | 66 | 3 | 63 |
| 4 | 49 | 2 | 47 |
| 4.5 | 38 | 1 | 37 |
| 5 | 18 | 0 | 18 |

**Observation**

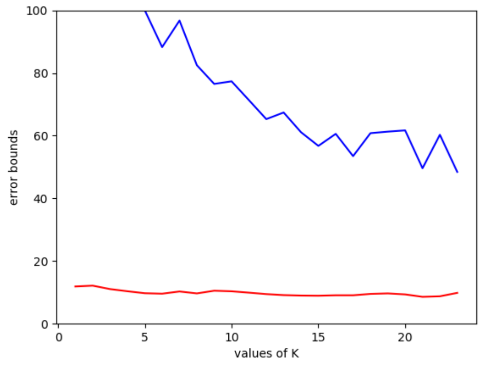
In the below results there are six different scenarios are described. Gradually we are increasing the distance between the mean of two spherical type clusters. Then for each case we calculated error bounds both theoretical and experimental. value of $k$ lies between $[1, \sqrt{n}]$. It can be observed that by increasing the distances between means of two spherical clusters the deviation of bounds are getting more tighter.

The reason behind this is **class overlapping data sets**. If two class means are more away then data set is easily separable. If separability is high means the classifier can be more accurately built. So, this will create less errors. But if class means are more closer then classifier can not works so well. Then error rate will be high.

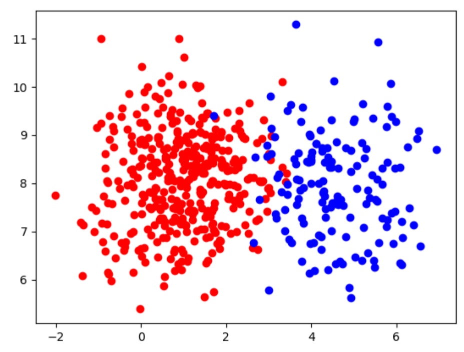
Another thing to notice that higher the distance between means the error rate is going to low to lower. This is because more the distance lesser the size of overlapping of two classes. If size of the overlapping region grows then misclassification region also grows. greater size of misclassification region more will be the error rate. If misclassification region size is less then error rate will decreases. That's the reason the experimental error rate gradually goes to zero.



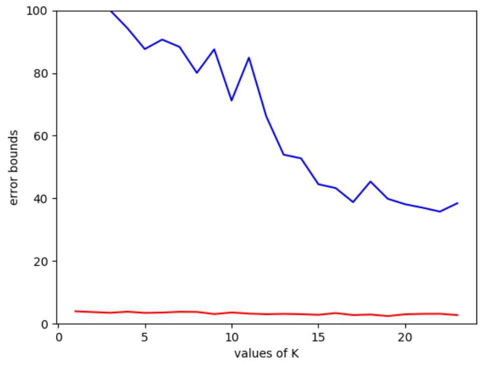
((c) ) Random Data 2



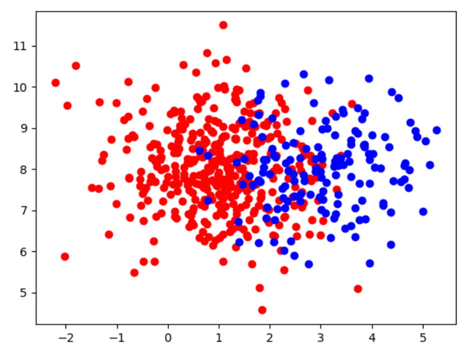
((d) ) Plot 2



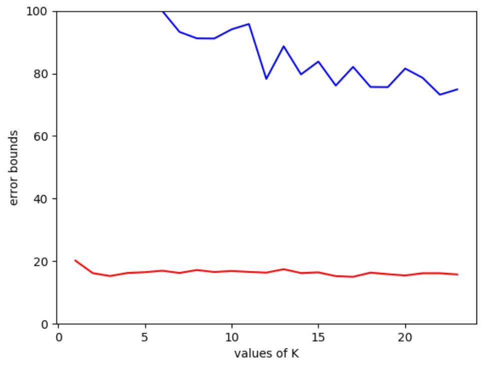
((e) ) Random Data 3



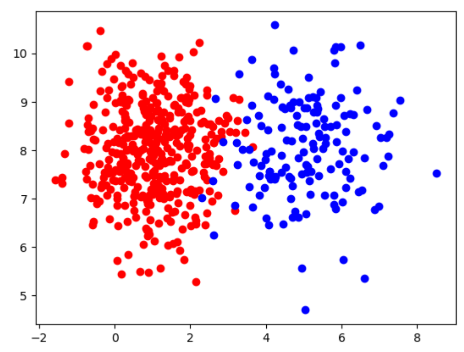
((f) ) Plot 3



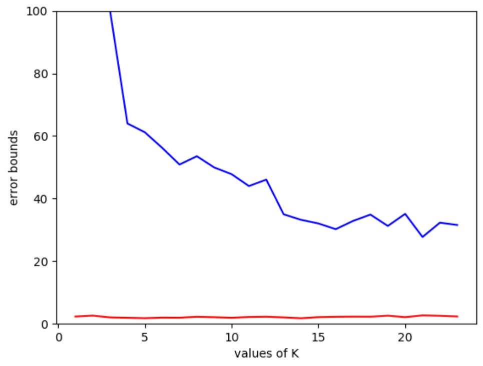
((a) ) Random Data 1



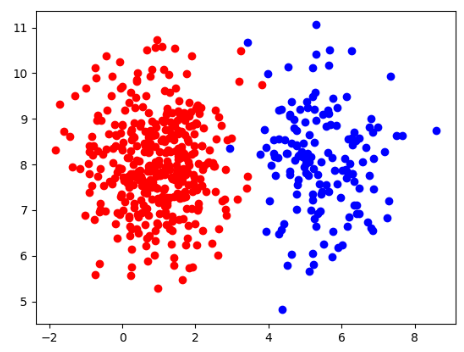
((b) ) Plot 1



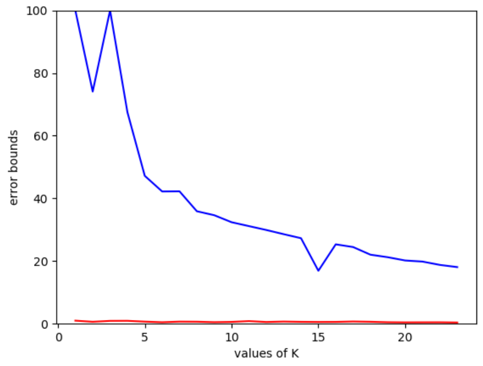
((g) ) Random Data 4



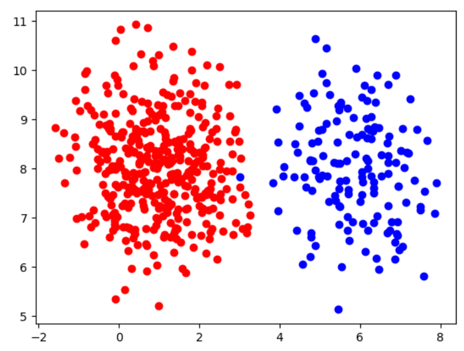
((h) ) Plot 4



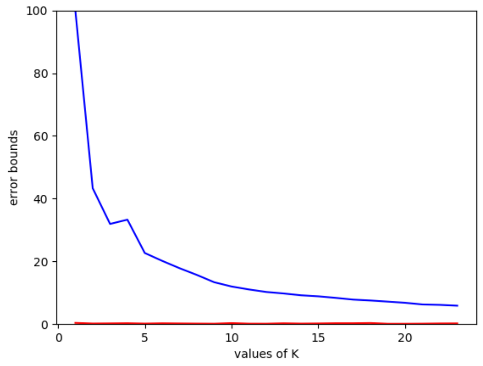
((i) ) Random Data 5



((j) ) Plot 5



((k) ) Random Data 6



((l) ) Plot 6

🟥 Experimental Bound

🟦 $Theoretical Bound$

Figure 6.3: Bound Validation for Overlapping Classes

### 6.3.4 Bound Validation for Range of $k$

In this stage we will check bound validation for a whole range of $k$ values. Here we used range as $k \in [1, \sqrt{n}]$. Refer Table 6.5 for data set details.

Table 6.5: KEEL Real World Data Sets

| No | Name | Samples | Features | Classes |
|---|---|---|---|---|
| 1 | Bupa | 345 | 6 | 2 |
| 2 | Haberman | 306 | 3 | 2 |
| 3 | Heart | 270 | 13 | 2 |
| 4 | Spectfheart | 267 | 44 | 2 |
| 5 | Sonar | 208 | 60 | 2 |
| 6 | Wine | 178 | 13 | 3 |
| 7 | Glass | 214 | 9 | 7 |
| 8 | Ecoli | 336 | 7 | 8 |
| 9 | Led7digit | 500 | 7 | 10 |
| 10 | Hayes-Roth | 132 | 4 | 3 |
| 11 | Tae | 151 | 5 | 3 |

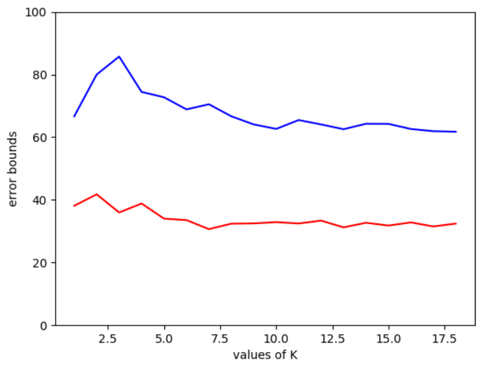
For results and bound deviation Refer Table 6.6

Table 6.6: Results Obtained for Range of $k$

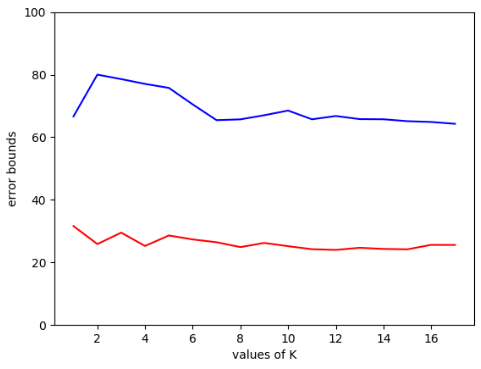
| No | Name | Th_bound | Exp_Bound | Deviation |
|---|---|---|---|---|
| 1 | Bupa | 67.72 | 33.83 | 33.89 |
| 2 | Haberman | 69.03 | 26.09 | 42.94 |
| 3 | Heart | 69.49 | 32.94 | 36.45 |
| 4 | Spectfheart | 68.32 | 25.0 | 43.32 |
| 5 | Sonar | 69.56 | 25.26 | 44.30 |
| 6 | Wine | 68.34 | 30.78 | 37.56 |
| 7 | Glass | 63.69 | 35.26 | 28.43 |
| 8 | Ecoli | 49.45 | 15.10 | 34.35 |
| 9 | Led7digit | 62.39 | 29.40 | 32.99 |
| 10 | Hayes-Roth | 66.73 | 41.73 | 25.00 |
| 11 | Tae | 67.29 | 57.74 | 9.55 |

#### Observation

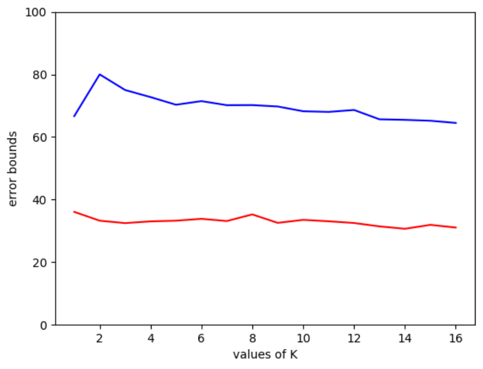
For bound validation consider below plots. Observe that theoretical error bound is always greater than experimental error bound. The monotonic nature of curve of theoretical bound varies with the value of $k$. So, theoretical bound has dependency on $k$ value. Experimental error bound varies with the change of $k$. In the initial stage the gap between theoretical bound and experimental bound i.e. bound deviation is higher. But after increasing of the value of $k$ the deviation gradually converges.



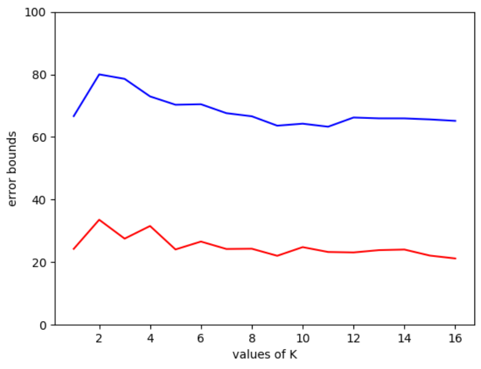
((a) ) Bupa



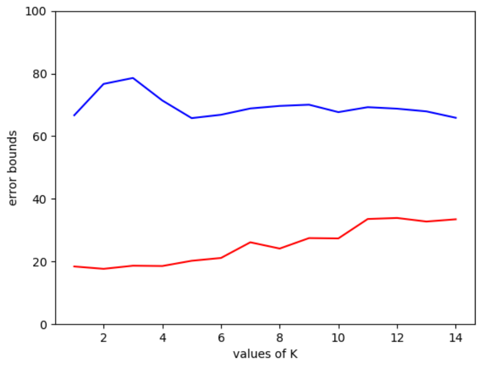
((b) ) Haberman



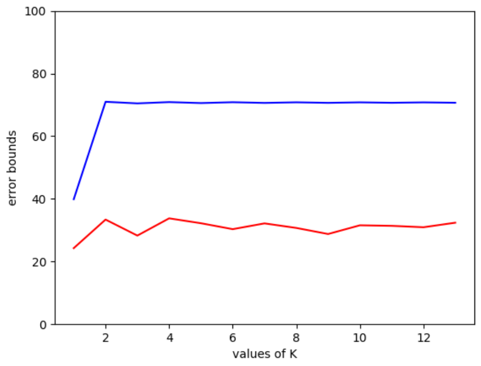
((c) ) Heart



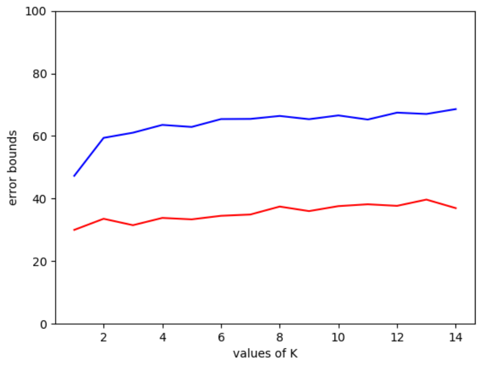
((d) ) Spectfheart



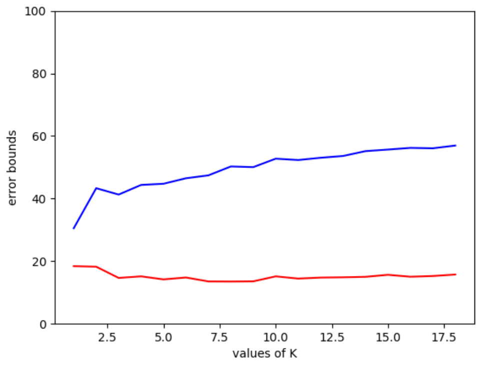
((e) ) Sonar



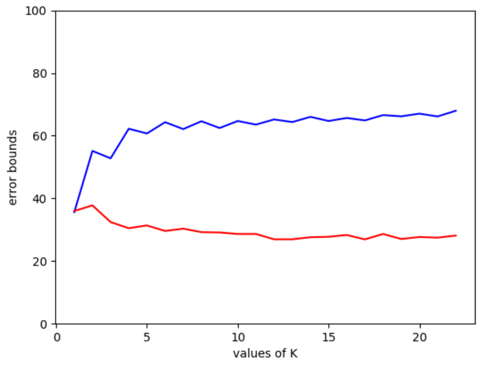
((f) ) Wine



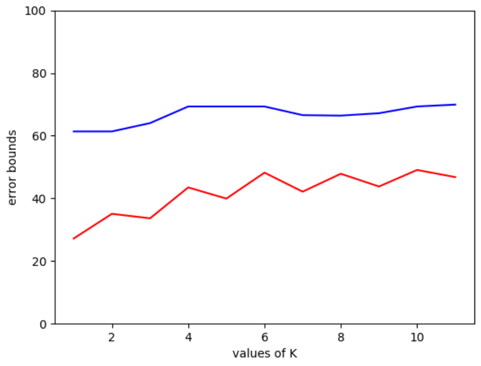
((g) ) Glass



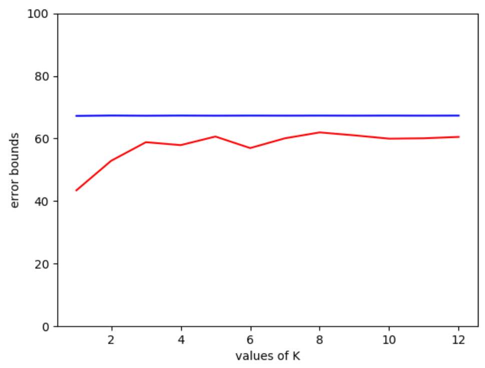
((h) ) Ecoli



((i) ) Led7digit



((j) ) Hayes-Roth



((k) ) Tae

🟥 Experimental Bound
🟦 Theoretical Bound

Figure 6.4: Bound Validation for Range of $k$ Values

## 6.4 Optimal Values of $k$

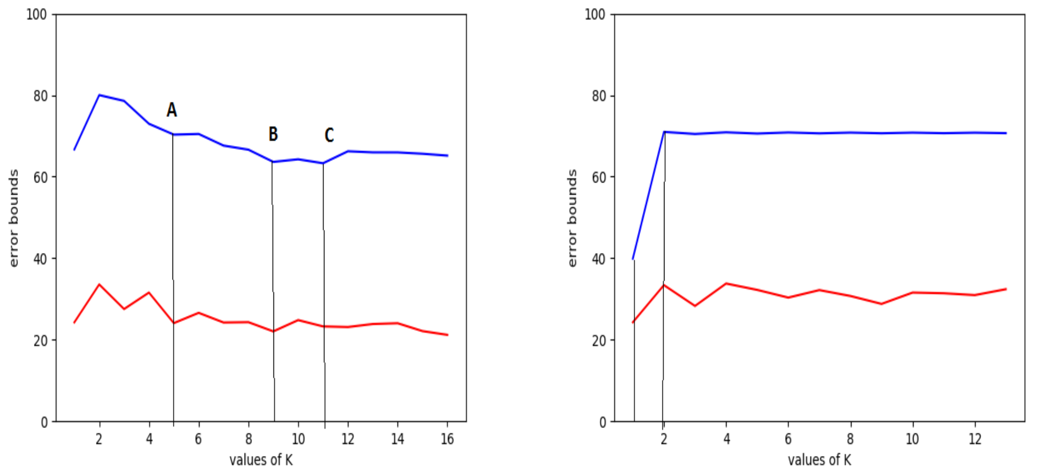
The main problem of using $k$-NN classifier is the value of $k$. While using $k$-NN algorithm but there is no idea what value of $k$ or set of values of $k$ to choose. So, we are keen to solve this problem. The solution will not provide a single optimal value of $k$ but it will generate a set(s) of $k$ values. If we select a value of $k$ from this set our error rate will be minimal. This is achieved by estimating error bound plots.

### Suggested Solution

- Plot the theoretical bound values with respect to values of $k$ where $k \in [1, \sqrt{n}]$.
- The theoretical error bound curve varies monotonic nature with change of value of $k$. So, for a fixed $k$ we can assure that error bound is fixed and experimental error bound will not cross this value.
- By observing the theoretical error bound curve, the curve sometimes becomes decreasing or sometimes increasing. So, it creates local minima. We can track for what value(s) of $k$ the curve attains local minima, this $k$ value will give minimal error rate. These set of values of are called **Optimal $k$ Values**.

### Example

We have a data set ”Spectfheart” which has 267 samples, 44 features, 2 class labels. After running the algorithm we got the following results



((a) ) Multiple Optimal Values ((b) ) Single Optimal Value

Figure 6.5: Set of Optimal $k$ Values

In the first example (see Figure 6.5) three local minima are identified as Point A, Point B, Point C. At this points we will get minimized error rate. Let $f(.)$ is the function which represents the theoretical error curve. Therefore,

$$f(k=5) \approx 69\%$$
$$f(k=9) \approx 62\%$$
$$f(k=11) \approx 61\%$$

From the above values we can see that minimal error rate is achieved at three points. Those $k$ values are $\{5, 9, 11\}$. So, with respect to the range $k \in [1, 16]$, the optimal set is $\{5, 9, 11\}$. The optimal set size can vary if the range of $k$ varies.

There can be another scenario if there is no local minima or having global minima at lower value of $k$. Then we can use that $k$ value also. Let see the second example (see Figure 6.5) which is generated by using a data set named ”Wine” having 178 samples, 13 features, 3 class labels and imbalance ratio is 1.5.

For $k = 1$ theoretical error bound is about 40%. But after $k \geq 2$ the error rate is fixed at approximately 70%. So, user can select $k = 1$ as its optimal choice. Here we have two optimal sets $\{1\}$ and $\{k | k \geq 2\}$

# Chapter 7

# Conclusion and Future Work

## 7.1 Conclusion

Our motto is to overcome the limitations of prevailing error bound algorithms. We have developed two different algorithms to estimate maximum probability of misclassification of an unlabeled test sample by using $k$-nearest neighbor classifier. This algorithms take only input data set, number of class labels and value of $k$. input data set can be any type of data set. Algorithms also work for two-class and multi-class data sets. Therefore, we overcome the challenges.

Our algorithms will also applicable to standard and imbalance data sets. It will detect the misclassification or overlapping region within the data set. Whatever may be the size of this region our proposed algorithm will detect properly. There can be many overlapping regions depending on number of class labels in the data set.This will also be detected.

We have performed different types of experiments and got expected results. There are very few data sets where bounds fails but almost all data sets showing satisfying result. We can get estimate of bound on probability of misclassification which has high importance in real life. In many cases we have a data set. We can not visualize data in higher dimensional feature space. So, there is no idea how the $k$-NN will behave on that data set but now our algorithms will come to rescue.

We also solves the problem of choosing value of $k$. Plotting of theoretical bound with $k$ values the local minima will be optimal $k$ values. So, using those set of $k$ values will lead to better accuracy.

Our implemented codes can be found in github link. The code is fully written using python programming language. In github repository the ReadMe file provides guidance to utilize the code for the users' own data sets. The link is given below

https://github.com/kushalbose92/error-bound-knn

## 7.2 Scope for Future Work

The algorithms are working finely but still it has some problems which are listed below. This problems can be assumed as our future work.

**Tight The Bound :** In previous chapter the several experiments are shown. Some data sets performs well but some are not satisfactory.Now our first target should be to reduce the bound deviation which is $(Th\_Bound - Exp\_Bound)$. This should be more tighter. Otherwise the algorithm will not be meaningful to use.

**Reduce Time Complexity :** As discussed in complexity analysis part that time complexity is dependent to number of samples in the data set. So, if number of samples becomes higher then expected completion of time will also be much higher. In real life scenario long duration is not desirable. In the section 5.4 one solution already discussed to handle large data sets. But results will be approximate. So, it is a major challenge to reduce the time complexity of our algorithms.

# Bibliography

[1] Thomas M Cover, Peter E Hart, et al. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[2] Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. Knn model-based approach in classification. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 986–996. Springer, 2003.

[3] Gautam Bhattacharya, Koushik Ghosh, and Ananda S Chowdhury. An affinity-based new local distance function and similarity measure for knn algorithm. *Pattern Recognition Letters*, 33(3):356–363, 2012.

[4] José Salvador Sánchez, Ramón Alberto Mollineda, and José Martínez Sotoca. An analysis of how training data complexity affects the nearest neighbor classifiers. *Pattern Analysis and Applications*, 10(3):189–201, 2007.

[5] Alexander Hinneburg, Charu C Aggarwal, and Daniel A Keim. What is the nearest neighbor in high dimensional spaces? In *26th Internat. Conference on Very Large Databases*, pages 506–515, 2000.

[6] Inderjeet Mani and I Zhang. knn approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of workshop on learning from imbalanced datasets*, volume 126, 2003.

[7] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[8] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009.

[9] Sahibsingh A Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, (4):325–327, 1976.

[10] James M Keller, Michael R Gray, and James A Givens. A fuzzy k-nearest neighbor algorithm. *IEEE transactions on systems, man, and cybernetics*, (4):580–585, 1985.

[11] Vincent Garcia, Eric Debreuve, and Michel Barlaud. Fast k nearest neighbor search using gpu. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–6. IEEE, 2008.

[12] Yi-Ching Liaw, Maw-Lin Leou, and Chien-Min Wu. Fast exact k nearest neighbors search using an orthogonal search tree. *Pattern Recognition*, 43(6):2351–2358, 2010.

[13] Eric Bax. Validation of $k$-nearest neighbor classifiers. *IEEE Transactions on Information Theory*, 58(5):3225–3234, 2011.

[14] Yunwen Lei, Ürün Dogan, Ding-Xuan Zhou, and Marius Kloft. Data-dependent generalization bounds for multi-class classification. *IEEE Transactions on Information Theory*, 2019.

[15] Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media, 2013.

[16] Jesús Alcalá-Fdez, Alberto Fernández, Julián Luengo, Joaquín Derrac, Salvador García, Luciano Sánchez, and Francisco Herrera. Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic & Soft Computing*, 17, 2011.

[17] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.