**Modified KNN Analysis**

**K-Value: K Nearest Neighbors.**

**T-Value: Number of Tuples from each class which will be used to calculate distance from test data.**

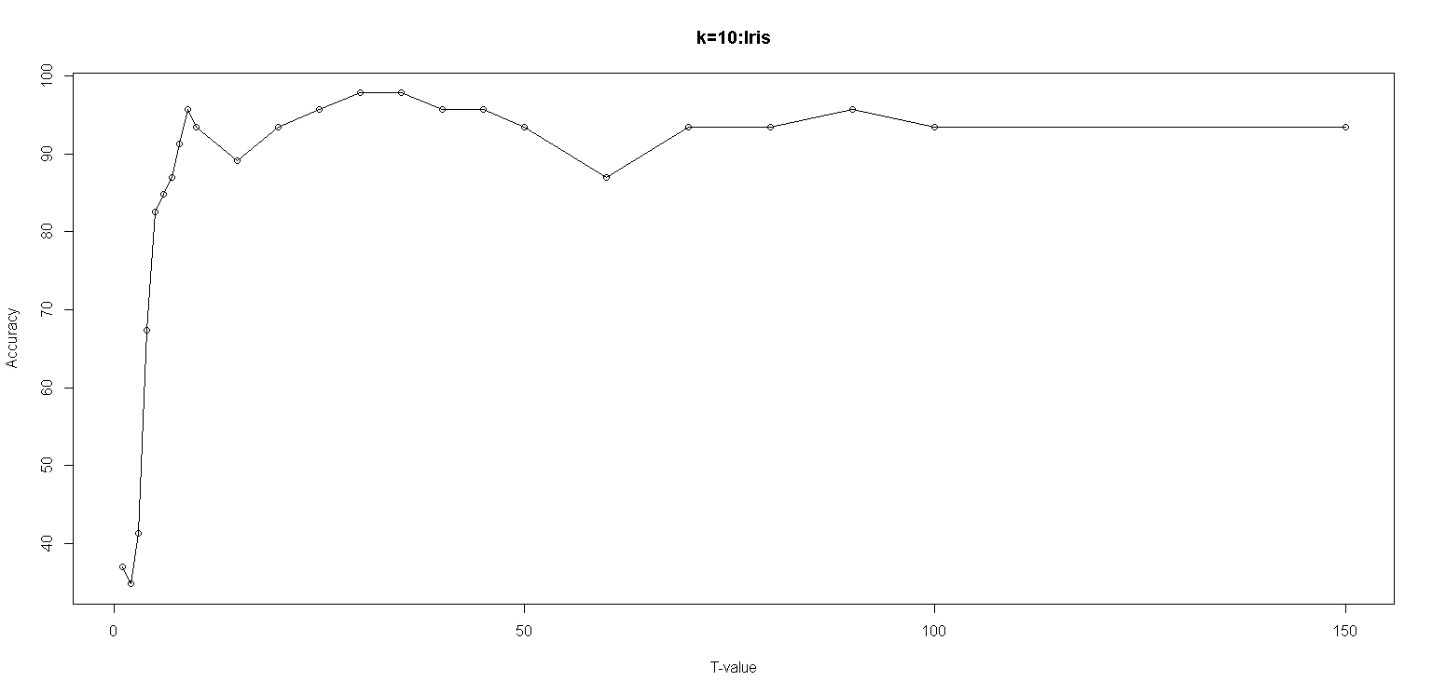
**Accuracy: (TRUE Predictions/Total Predictions)\*100.**

**Modification:**

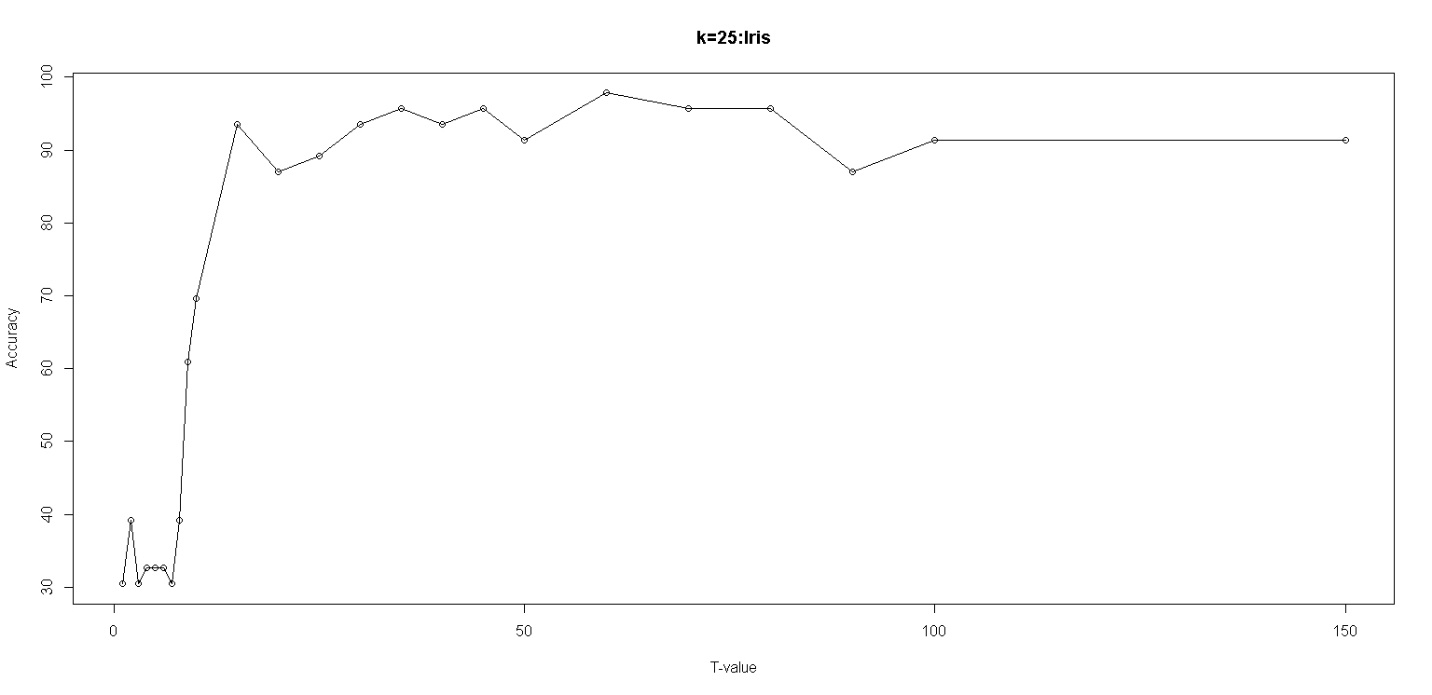
**Modified KNN covers the ability of simple KNN. Moreover, it adds extra features. Instead of calculating distances with all the train data elements, modified\_knn extracts ‘t’ tuples from each class from train data set. The distance of test data is calculated with these tuples.**

**DATASET: Iris**

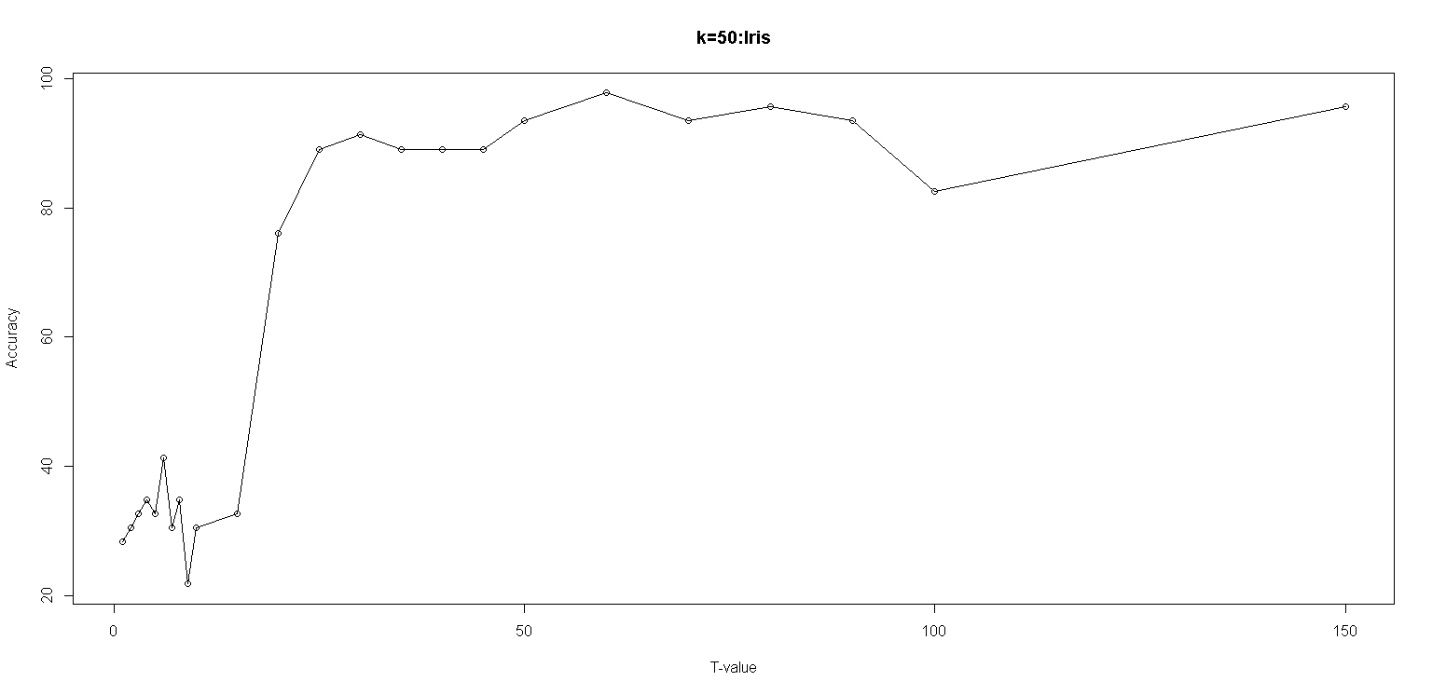
|  |  |  |
| --- | --- | --- |
| K-Value | T-Value | Accuracy |
| 10 | 1 | 36.95652 |
| 10 | 2 | 34.78261 |
| 10 | 3 | 41.30435 |
| 10 | 4 | 67.39130 |
| 10 | 5 | 82.60870 |
| 10 | 6 | 84.78261 |
| 10 | 7 | 86.95652 |
| 10 | 8 | 91.30435 |
| 10 | 9 | 95.65217 |
| 10 | 10 | 93.47826 |
| 10 | 15 | 89.13043 |
| 10 | 20 | 93.47826 |
| 10 | 25 | 95.65217 |
| 10 | 30 | 97.82609 |
| 10 | 35 | 97.82609 |
| 10 | 40 | 95.65217 |
| 10 | 45 | 95.65217 |
| 10 | 50 | 93.47826 |
| 10 | 60 | 86.95652 |
| 10 | 70 | 93.47826 |
| 10 | 80 | 93.47826 |
| 10 | 90 | 95.65217 |
| 10 | 100 | 93.47826 |
| 10 | 150 | 93.47826 |



|  |  |  |
| --- | --- | --- |
| K-Value | T-Value | Accuracy |
| 25 | 1 | 30.43478 |
| 25 | 2 | 39.13043 |
| 25 | 3 | 30.43478 |
| 25 | 4 | 32.60870 |
| 25 | 5 | 32.60870 |
| 25 | 6 | 32.60870 |
| 25 | 7 | 30.43478 |
| 25 | 8 | 39.13043 |
| 25 | 9 | 60.86957 |
| 25 | 10 | 69.56522 |
| 25 | 15 | 93.47826 |
| 25 | 20 | 86.95652 |
| 25 | 25 | 89.13043 |
| 25 | 30 | 93.47826 |
| 25 | 35 | 95.65127 |
| 25 | 40 | 93.47826 |
| 25 | 45 | 95.65217 |
| 25 | 50 | 91.30435 |
| 25 | 60 | 97.82609 |
| 25 | 70 | 95.65217 |
| 25 | 80 | 95.65217 |
| 25 | 90 | 86.9562 |
| 25 | 100 | 91.30435 |
| 25 | 150 | 91.30435 |

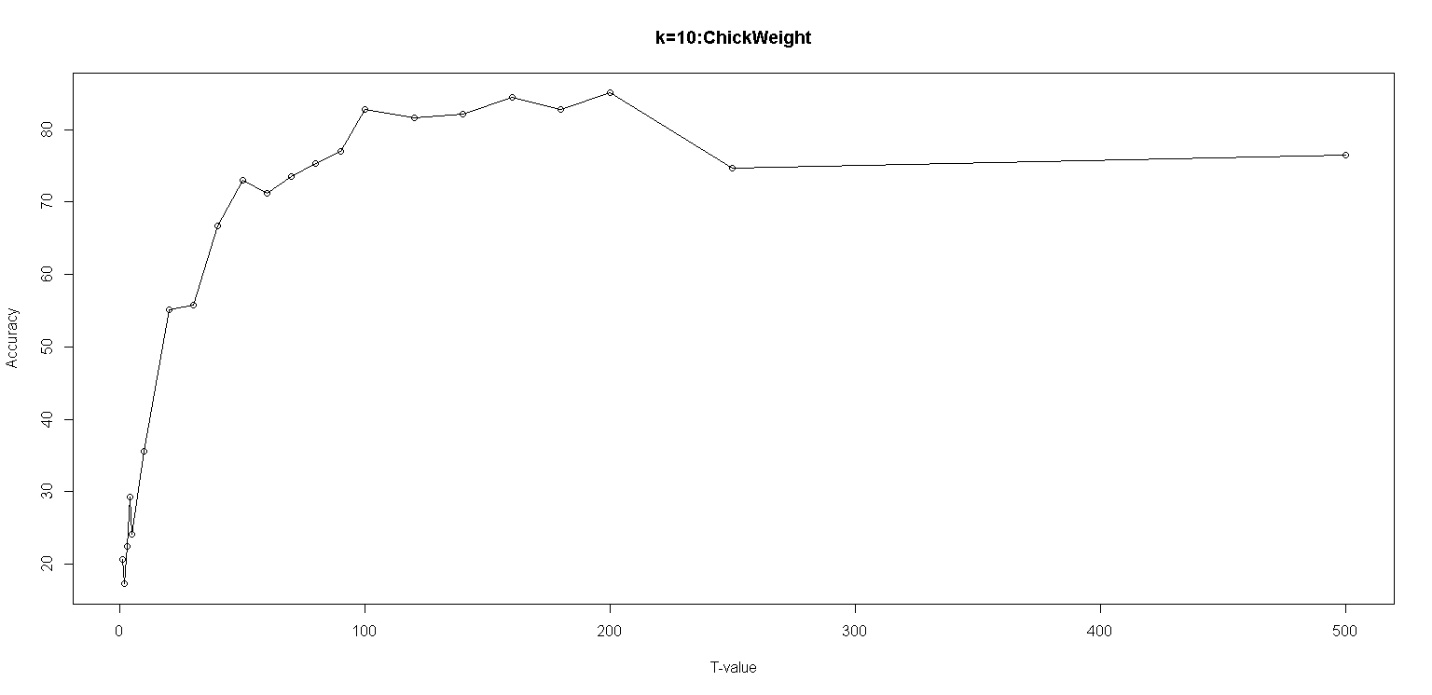


|  |  |  |
| --- | --- | --- |
| K-Value | T-Value | Accuracy |
| 50 | 1 | 28.26087 |
| 50 | 2 | 30.43478 |
| 50 | 3 | 32.60870 |
| 50 | 4 | 34.78261 |
| 50 | 5 | 32.60870 |
| 50 | 6 | 41.30435 |
| 50 | 7 | 30.43478 |
| 50 | 8 | 34.78261 |
| 50 | 9 | 21.73913 |
| 50 | 10 | 30.43478 |
| 50 | 15 | 32.60870 |
| 50 | 20 | 76.08696 |
| 50 | 25 | 89.13043 |
| 50 | 30 | 91.30435 |
| 50 | 35 | 89.13043 |
| 50 | 40 | 89.13043 |
| 50 | 45 | 89.13043 |
| 50 | 50 | 93.47826 |
| 50 | 60 | 97.82609 |
| 50 | 70 | 93.47826 |
| 50 | 80 | 95.65217 |
| 50 | 90 | 93.47826 |
| 50 | 100 | 82.60870 |
| 50 | 150 | 95.65217 |

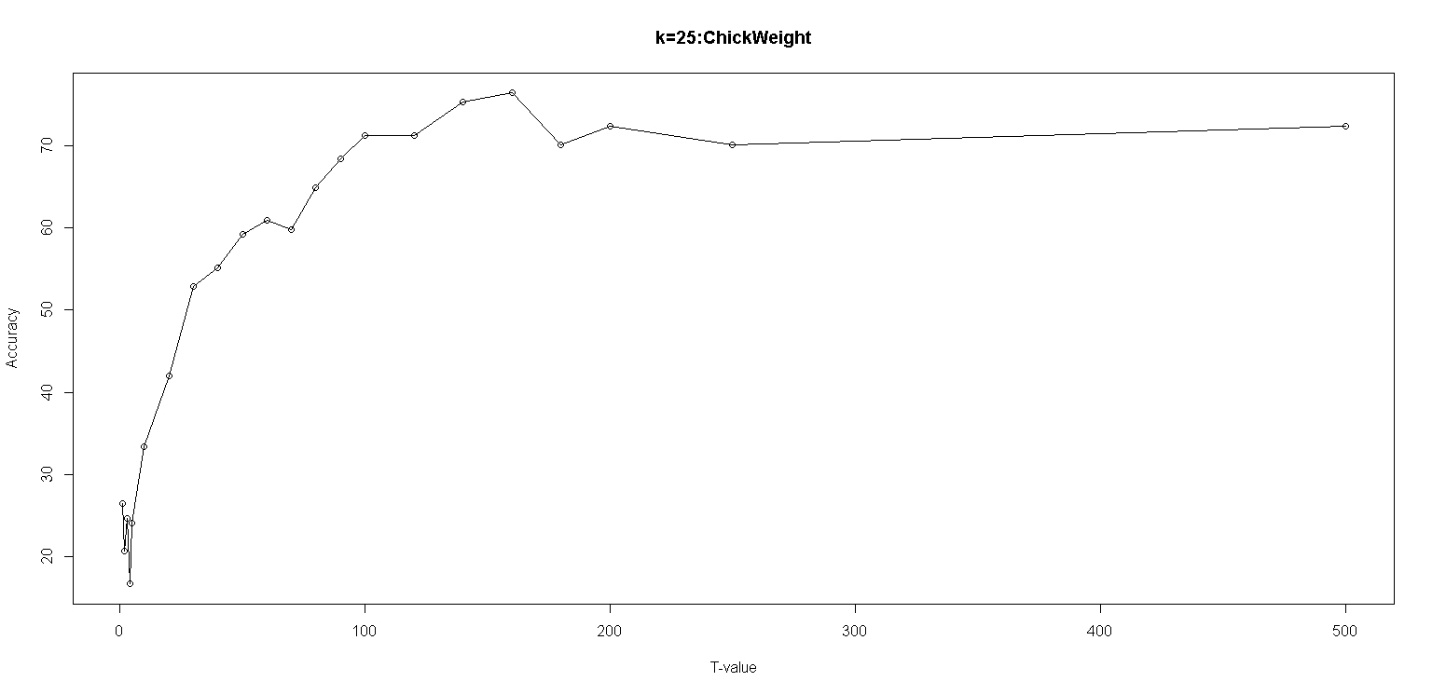


**DATASET: ChickWeight**

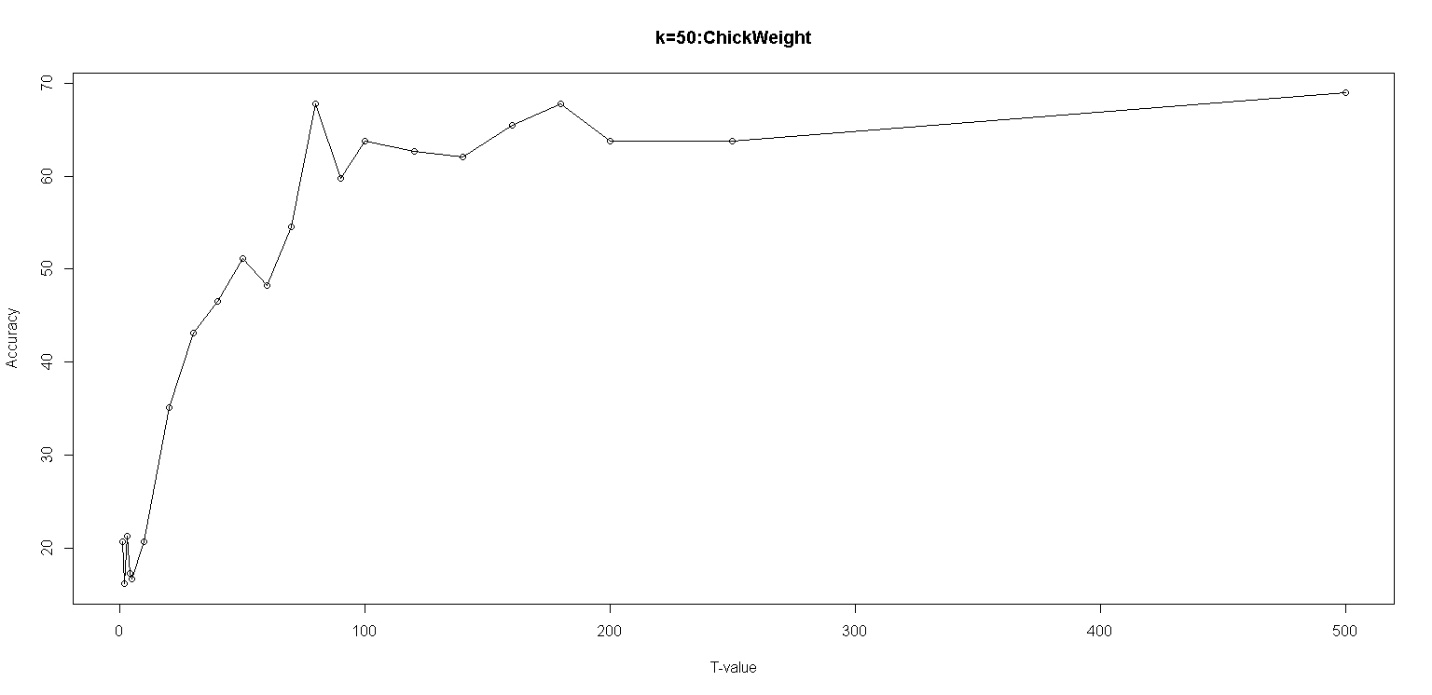
|  |  |  |
| --- | --- | --- |
| K-Value | T-Value | Accuracy |
| 10 | 1 | 20.68966 |
| 10 | 2 | 17.24138 |
| 10 | 3 | 22.41739 |
| 10 | 4 | 29.31034 |
| 10 | 5 | 24.13793 |
| 10 | 10 | 35.63218 |
| 10 | 20 | 55.17241 |
| 10 | 30 | 55.74713 |
| 10 | 40 | 66.66667 |
| 10 | 50 | 72.98891 |
| 10 | 60 | 71.26437 |
| 10 | 70 | 73.56322 |
| 10 | 80 | 75.28736 |
| 10 | 90 | 77.01149 |
| 10 | 100 | 82.75862 |
| 10 | 120 | 81.60920 |
| 10 | 140 | 82.18391 |
| 10 | 160 | 84.48276 |
| 10 | 180 | 82.75862 |
| 10 | 200 | 85.05747 |
| 10 | 250 | 74.71264 |
| 10 | 500 | 76.43678 |



|  |  |  |
| --- | --- | --- |
| K-Value | T-Value | Accuracy |
| 25 | 1 | 26.43678 |
| 25 | 2 | 20.68966 |
| 25 | 3 | 24.71264 |
| 25 | 4 | 16.66667 |
| 25 | 5 | 24.13793 |
| 25 | 10 | 33.33333 |
| 25 | 20 | 41.95402 |
| 25 | 30 | 52.87356 |
| 25 | 40 | 55.17241 |
| 25 | 50 | 59.19540 |
| 25 | 60 | 60.91954 |
| 25 | 70 | 59.77011 |
| 25 | 80 | 64.94253 |
| 25 | 90 | 68.39080 |
| 25 | 100 | 71.26437 |
| 25 | 120 | 71.26437 |
| 25 | 140 | 75.28736 |
| 25 | 160 | 76.43678 |
| 25 | 180 | 70.11494 |
| 25 | 200 | 72.41379 |
| 25 | 250 | 70.11494 |
| 25 | 500 | 72.41379 |

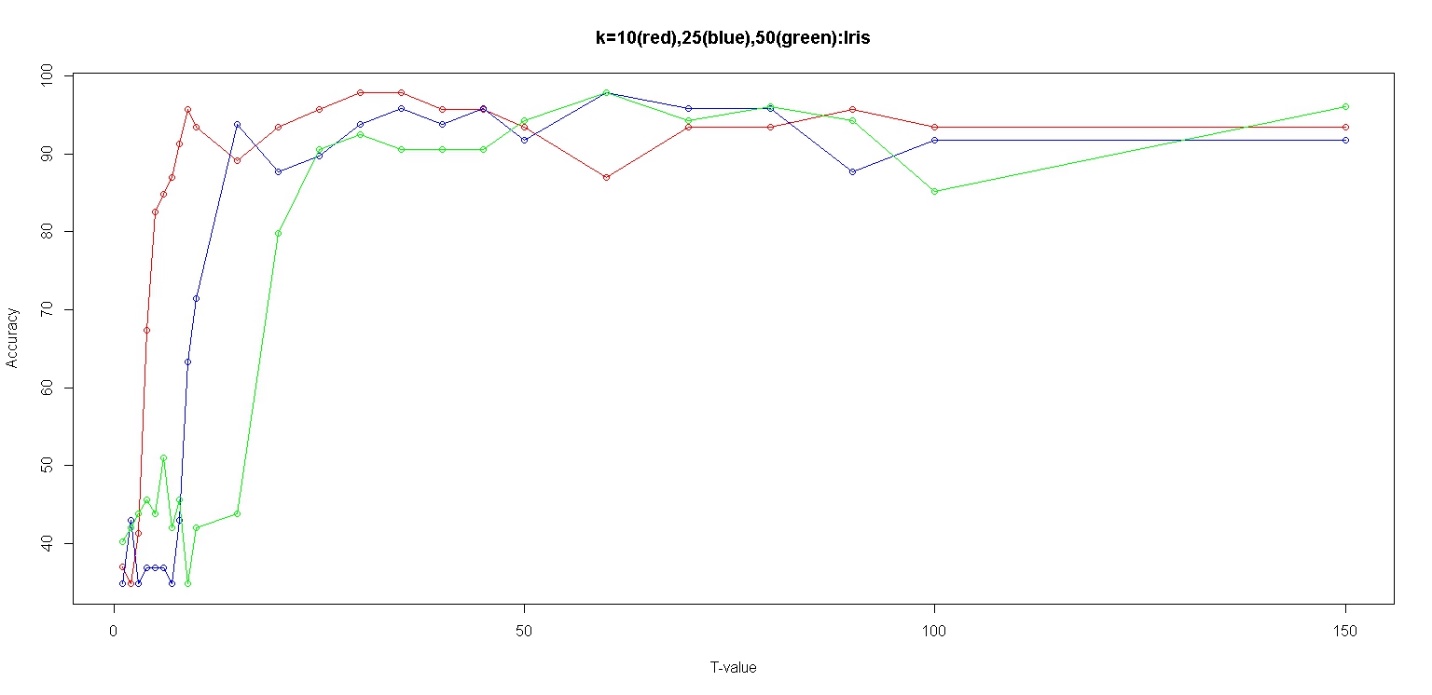


|  |  |  |
| --- | --- | --- |
| K-Value | T-Value | Accuracy |
| 50 | 1 | 20.68966 |
| 50 | 2 | 16.09195 |
| 50 | 3 | 21.26437 |
| 50 | 4 | 17.24138 |
| 50 | 5 | 16.66667 |
| 50 | 10 | 20.68966 |
| 50 | 20 | 35.05747 |
| 50 | 30 | 43.10345 |
| 50 | 40 | 46.55172 |
| 50 | 50 | 51.14943 |
| 50 | 60 | 48.27586 |
| 50 | 70 | 54.59770 |
| 50 | 80 | 67.81609 |
| 50 | 90 | 59.77011 |
| 50 | 100 | 63.79310 |
| 50 | 120 | 62.64368 |
| 50 | 140 | 62.06897 |
| 50 | 160 | 65.51274 |
| 50 | 180 | 67.81609 |
| 50 | 200 | 63.79310 |
| 50 | 250 | 63.79310 |
| 50 | 500 | 68.96552 |

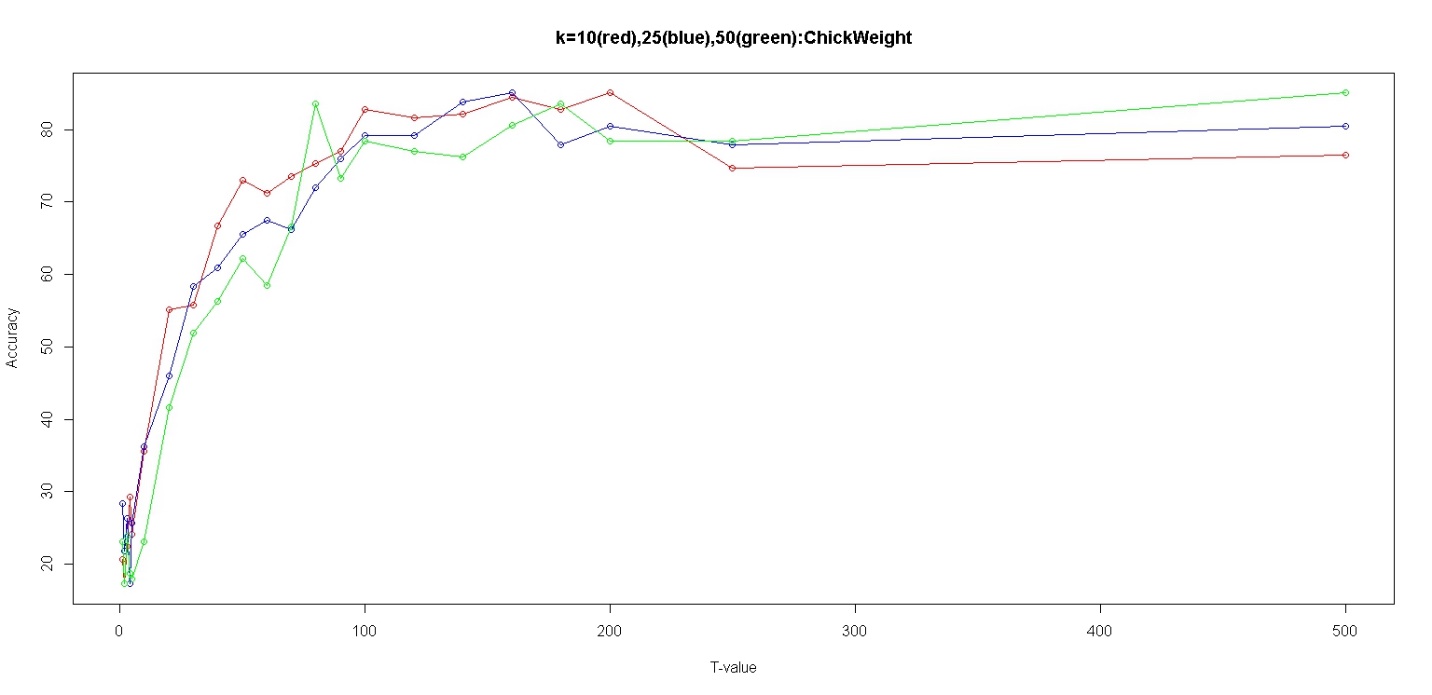


**COMBINED GRAPH ANALYSIS**

1. **IRIS**



1. **CHICKWEIGHT**

****

**Advantages over simple KNN:**

1. **Proper Handling of Biased Dataset:**

**Suppose, a data set has 1000 tuples and 3 classes.**

**500 tuples belong to class1.**

**400 tuples belong to class2.**

**100 tuples belong to class3.**

**When a tuple from test data set is computed for distances with all 1000 tuples in training data set. Biasing is most probable.**

**In Modified\_KNN, if we select T-Value = 50.**

**50 tuples from each of the classes i.e. class1, class2, class3 will be extracted at random and with these tuples the test data distance computation will take place.**

**This approach eliminates possibility of biasing and sheds improved results.**

1. **Reduced Time Complexity:**

**From the graphs above, it can be inferred that accuracy gets saturated after a threshold value is met.**

**There is no need to process entire train data set for each tuple. Less number of tuples give the same accurate results.**

1. **Reduced Space Complexity:**

**Obviously, if we process less number of tuples, temporary storage requirement also reduces proportionally. Therefore, Space Complexity is also reduced.**

**Code:**

knn.df <- read.csv('D:\\Study\\Projects\\AD\\iris-species\\Iris.csv', header = FALSE, sep = ',')

class\_labels<-c("Iris-setosa","Iris-versicolor","Iris-virginica")

#chick.df<- ChickWeight

#class\_labels<-c(1,2,3,4)

Mode = function(x){

ta = table(x)

tam = max(ta)

mod = 0

if (all(ta == tam))

mod = names(ta)[ta == tam]

else

if(is.numeric(x))

mod = as.numeric(names(ta)[ta == tam])

else

mod = names(ta)[ta == tam]

return(mod)

}

#my knn

#data : datase,t: no of tuples to extract, class\_labels: vector\_to\_store\_labels,col: column\_no having class\_values

restricted\_knn <- function(data,k,t,class\_lablels,col){

#sampling data & constructing train and test datasets

data<- data[sample(nrow(data)),]

train\_data <- data[1:as.integer(0.7\*nrow(data)),]

test\_data <- data[as.integer(0.7\*nrow(data) +1):nrow(data),]

#predicted class values for test\_data

pred\_col <- c()

#loop runs for each tuple in test\_data

for(i in c(1:nrow(test\_data))){

#different class data\_sets

a<-data.frame()

#extract t tuples of each class from train\_data

for(x in c(1:length(class\_labels)))

{

temp <- train\_data[train\_data[col] == class\_labels[x],]

if(nrow(temp) > t)

temp <- temp[sample(nrow(temp)),]

temp <- temp[c(1:t),]

a<-rbind(a,temp)

}

#distance: distance b/w cur\_test\_data and all\_train\_data

distance <- c()

#class: respective train\_data classes

class <- c()

for(j in 1:nrow(a)){

#removing the col that contains class values

v<- c(1:ncol(a))

v<- v[-col]

m <- matrix(c(test\_data[i,v],a[j,v]),ncol=ncol(a)-1,byrow=TRUE)

distance <- c(distance,dist(m,method="euclidean"))

class <- c(class,as.character(a[j,][[col]]))

}

#merging distance and class in one data frame and then sorting wrt distance in ascending order

eu <- data.frame(distance,class)

eu <- eu[order(eu$distance),]

#selecting top k distances and their classes

if(k < nrow(eu))

eu <- eu[1:k,]

pred\_col <- c(pred\_col,max(Mode(eu$class)))

}

test\_data["PREDICTED\_CLASS"] <- NA

test\_data$PREDICTED\_CLASS <- pred\_col

test\_data["JUDGEMENT"] <- as.logical(test\_data[col] == test\_data$PREDICTED\_CLASS)

return(test\_data)

}

#col1 actual value column, col2 predicted value column

accuracy <- function(test\_data,col1,col2){

correct = 0

for(i in c(1:nrow(test\_data))){

if(test\_data[i,col1] == test\_data[i,col2]){

correct = correct+1

}

}

accu = correct/nrow(test\_data) \* 100

return(accu)

}

#tvec for ChickWeight

#tvec<-c(1,2,3,4,5,10,20,30,40,50,60,70,80,90,100,120,140,160,180,200,250,500)

#tvec for Iris

tvec<-c(1,2,3,4,5,6,7,8,9,10,15,20,25,30,35,40,45,50,60,70,80,90,100,150)

analysis<-function(data,col1,col2,k,tvec)

{

c1<-rep(k,length(tvec))

c3<-c()

for(i in c(1:length(tvec)))

{

a<-restricted\_knn(data,k,tvec[i],class\_labels,col1)

ac<-accuracy(a,col1,col2)

c3<-c(c3,ac)

}

pp<-cbind(c1,tvec,c3)

return(pp)

}

#PLOT GRAPHS

#plot(tvec,ansIris10[,3],main="k=10:Iris",xlab="T-value",ylab="Accuracy",type="o")