Homework 2

***Anushka Nair, Bam Nakapakorn, and Madeline Luong***

*Study Datasets:* BELL, CHILLER, and FREESTYLE fonts from given website.

BELL contains 956 observations/ instances and 412 features.

CHILLER contains 962 observations/ instances and 412 features.

FREESTYLE contains 956 observations/ instances and 412 features.

*Preliminary treatment:*

Unneeded excess features, non-numerical values, and non- defined classes discarded.

BELL\_CLEAN contains 239 observations/ instances and 404 features.

CHILLER\_CLEAN contains 238 observations / instances and 404 features.

FREESTYLE\_CLEAN contains 239 observations/ instances and 404 features.

Then defining the three classes (CL1, CL2. And CL3) to each dataset and unionizing them into a full dataset.

DATA then contains 716 observations / instances and 404 features.

*Part 0*

Computing the mean(where m1 = mean(x1)…mean(400)=m400) and standard deviation (s1= sd(x1)..sd(x400) =sd400) of the full data set. Here we standardize and scale the data. We want to standardize the features to the center at 0 and a standard deviation of 1 because different variables can be measured at different scales leading to bias.

We can see here that the standard deviation values are not one, so we will then normalize the data by the function yj = (xj – mj)/ sj. After standardizing the data, a columns standard deviation will be extracted. Column 3’s standard deviation is one, confirming that the data set has been rescaled and standardized, which we then name SDATA. To view the table containing all means and standard deviation prior to the standardization refer to the appendix section.

We then compute the correlation matrix of the 400 features. Finding the correlation matrix will show us the correlation coefficients between the 400 features. We want to display the ten highest absolute correlation coefficient values (shown in table below). A correlation matrix only calculates the correlation between numeric features, so we took out all non-numeric data, leaving us with a 400 x 400 array.

|  |  |  |
| --- | --- | --- |
| **Pixel Positions** | | **Correlation Coefficient** |
| **Xi, Xj, Pair** | |
| r19c16 | r19c15 | 0.91627 |
| r0c4 | r0c3 | 0.91518 |
| r15c16 | r14c16 | 0.91107 |
| r19c15 | r19c14 | 0.90619 |
| r15c17 | r14c17 | 0.90612 |
| r14c18 | r13c18 | 0.90470 |
| r11c18 | r10c18 | 0.90164 |
| r0c5 | r0c4 | 0.89906 |
| r12c1 | r11c1 | 0.89357 |
| r0c3 | r0c2 | 0.88880 |

*Part 1*

1. *- Creating a TRAINSET and TESTSET*

Taking the SDATA set, we will classify CL to SETROW, where CL1 is equal to SETROW1, CL2 is equal to SETROW2, and CL3 is equal to SETROW3. We then separate each class into individual subsets of SETROW1, SETROW2, and SETROW3 to create the proper train and test set distribution. Each subset will then be split arbitrarily where TRAIN will be about 80% of the subset and TEST will be about 20% of the subset. After each subset has been split into train and test sets, all train sets will be unionized into a full TRAINSET (trainsetcl1, trainsetcl2, and trainset cl3). The same will be done to the TESTSET (testsetcl1, testsetcl2, and testsetcl3).

1.1 - *Using the K- Nearest Neighbor algorithm (KNN) on the classification of CL1, CL2 , CL3.*

To apply the KNN algorithm, a matrix with the predictors in the TRAINSET is created labeled TRAIN\_no. Similarly, a matrix containing the predictors in the TESTSET is created labeled

TEST\_no. Then two vectors are created that contain the class labels for the training observations and testing observations which are labeled TRAIN\_label and TEST\_label respectively. We then apply the knn() function on the TRAINSET and TESTSET to predict the CL classifications of each font based on grey level image intensity for pixel at each position. We set a random seed before applying knn() because if several observations are tied as nearest neighbor then R will break the tie. Here we used k = 12.

Percent correct classifications

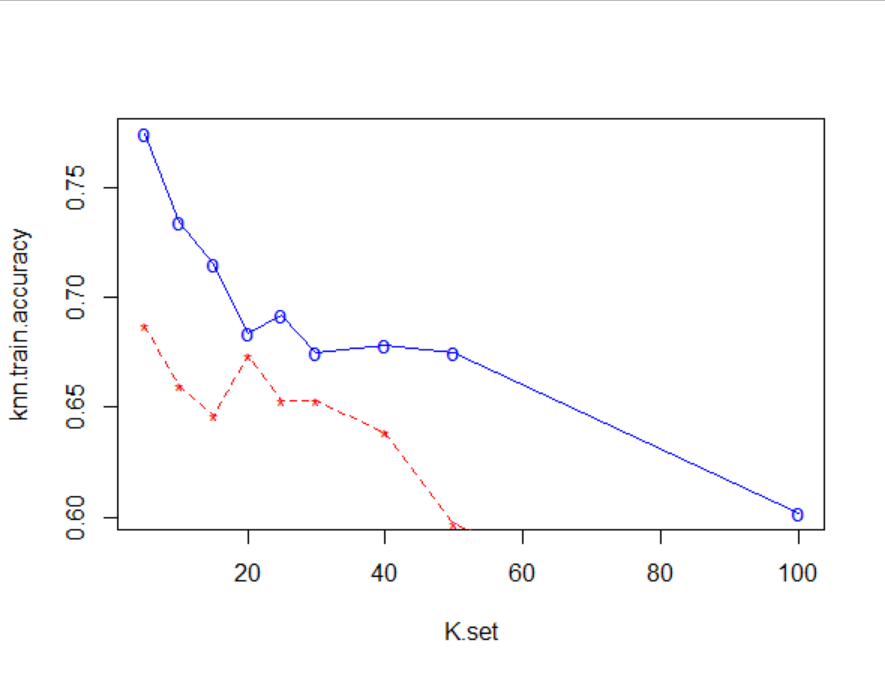
*Trainperf12* = 0.733

*Testperf12* = 0.687

We can see that the testperf accuracy is lower compared to trainperf. This makes sense because in the case of trainperf, the knn algorithm is trained and tested on the same data, while in the case of testperf, the algorithm has not been exposed to the data it is testing. Additionally, our test set is too small to conclude that it would perform similarly on new data. As we change the k value, the train set error could decline, however the test error may not. As we increase the K value, the variance will decrease but there will be a higher bias in classifier. If we decrease the K-value we might overfit the boundaries. So further exploration is needed to understand if increasing or decreasing the k value will decrease the error rate.

1.2 - *Replicating KNN algorithm for K = 5,10,15,20,30,40,50 and 100 and visualizing it.*

By using the same prior KNN function, to replicate this for values 5,10,15,20,30,40, 50 and

100 a loop will be created to identify the percent accuracy. This will be done to both the train and the test set. Then plotted to visualize the best range of k value. The plot of the k values will indicate if the values will overfit between each set.

Percent accuracy of each K value *Testperf5=* 68.75%

*Testperf10=* 66.66%

*Testperf15=* 66.66%

*Testper20=* 68.05%

*Testperf30=* 61.80%

*Testperf40=* 59.02%

*Testperf50=* 58.33%

*Testperf100=* 58.33%

Based on these values we can conclude

that where k is [5 < k < 20] is at least 68%

accuracy.

*Figure 1:plot of percent accuracy with varying k with both test and train sets.*

A plot showing a negative linear correlation of percent accuracy which is obtained by using k = 5,10, 15, 20,30,40,50

100. The blue line is the trainset and the red line is the testset values. Each dot indicates the correlated k value and accuracy.

According to the curves of train and test sets, we can see that we have slightly overfit model, where the test error is higher than the train error by (3%). The discrepancy between the 2 curves show the magnitude of overfit. Allowing us to select knn =5 as the best fit of accuracy.

1.3 - *Identifying the “best” k value within the prior range.*

From the prior loop, we will run it again using a sequence function from the range [5:100] in sets of

5. Thus, testing it on 5,10,15,20,25…100. Which we can conclude that the “ideal” k value will be 5 with

the percent accuracy of 68.75%

1.4 - *KNN algorithm using K = 5 and visualizing the respected correlation matrices and confidence intervals*

Based on the prior loop, we determine that k = 5. Then, we will run KNN algorithm on both the

TESTSET and TRAINSET. After receiving the accuracy values, we will then do a 3x3 confusion

matrix on both TESTSET and TRAINSEST to better visualize the classification for each class.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Train |  | PREDICTION | | |
|  | **Test K = 5** | **CL1** | **CL2** | **CL3** |
| TRUE | **CL1** | 71.63% | 20.00% | 8.37% |
| **CL2** | 14.63% | 80.49% | 4.88% |
| **CL3** | 6.74% | 7.77% | 85.49% |

A screenshot of a cell phone

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test |  | PREDICTION | | |
|  | **Test K = 5** | **CL1** | **CL2** | **CL3** |
| TRUE | **CL1** | 64.29% | 28.57% | 7.14% |
| **CL2** | 21.21% | 69.70% | 9.09% |
| **CL3** | 9.09% | 16.36% | 74.55% |

A screenshot of a cell phone

Description automatically generated

*Figure 2: Confusion matrices and percentage accuracy of TESTSET and TRAINSET*

Top left and bottom left are the confusion matrix for train set and test set, which the rows are the predictions and the columns are the actual values. Right side of the figure will be the corresponding accuracy percentage for each classifier.

*Train set*

Based on the figure, we can see that 154 instances of “CL1” are classified correctly as “CL1”. Then, 132

Instances of “CL2” are also classified correctly as “CL2”. “CL3” was classified correctly 165 instances.

61 instances of “CL1” were classified incorrectly as “CL2” or “CL3”. 32 instances of “CL2” were classified incorrectly as either “CL1” or “CL3”. Lastly, 28 instances of “CL3” were classified incorrectly as either “CL1” or “CL2”. Based on the percentages we can see that “CL3” was classified with the highest accuracy of 85.5%, comparatively to “CL2” at 80.5% and “CL1” at 71.6%.

*Testset*

Based on the figure, we can see that 36 instances of “CL1” are classified correctly as “CL1”. Then, 23

Instances of “CL2” are also classified correctly as “CL2”. “CL3” was classified correctly 41 instances.

20 instances of “CL1” were classified incorrectly as “CL2” or “CL3”. 10 instances of “CL2” were classified incorrectly as either “CL1” or “CL3”. Lastly, 14 instances of “CL3” were classified incorrectly as either “CL1” or “CL2”. Based on the percentages we can see that “CL3” was classified with the highest accuracy of 74.5% comparatively to “CL2” at 69.7% and “CL1” at 64.3%

Overall, we can see that “CL3” had the “best” prediction with the k value at 5.

1.5 - *Determining the confidence interval for both TESTSET and TRAINSET.*

After determining the confusion matrix, we seek to find the confidence intervals diagonals to determine they overlap or are nearly disjoint with each other.

*Confidence interval*

Trainset: 95% CI : (0.7531, 0.8199)

Testset: 95% CI : (0.6150, 0.7638)

We can see that the confidence intervals overlap with each other, which is evidence that quantities are

close to each other. However, we cannot conclude that trainset quantile is greater than testset quantile.

1.6 - Creating the individual packages from the standardize full data set (SDATA).

We going to separate the data into 4 packs (PACK1, PACK2, PACK3, and PACK 4). Each pack will have 100 features corresponded to the 100 pixel intensities. We will divide the pack by rLcM as follows:

PACK 1: L = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

M = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

PACK 2: L = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

M = 10, 11, 12, 13, 14, 15, 16, 17, 18, 19

PACK 3: L = 10, 11, 12, 13, 14, 15, 16, 17, 18, 19

M = 10, 11, 12, 13, 14, 15, 16, 17, 18, 19

PACK 4: L = 10, 11, 12, 13, 14, 15, 16, 17, 18, 19

M = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

For each pack, we create train (80%) and test (20%) subsets. Then we will apply KNN classification using Kbest = 5 to each pack and assign the accuracy percentage as weighted values w1, w2, w3, and w4.

Accuracy of PACK 1

*W1 = testperfbestk = 58%*

1.7 - Finding the accuracy for each PACK using KNN algorithm, where K= 5.

Using the same script from 1.6, replicating for the following PACKS.

Accuracy of PACKS

*W2 = testperfbestk = 75%*

*W3 = testperfbestk = 63%*

*W4 = testperfbestk = 66%*

According to the KNN classification to each pack, we can see that pack 2 has the highest accuracy rate of 75%. On the other hand, pack 1 resulted to be the lowest accuracy rate of 58%. This is 5 and 8 percentual points behind packs 3 and 4 respectively.

1.8 - Finding the KNN with the weighted features on global performance and TESTSET

We use the accuracy rates we found in 1.7 to weight each pack before standardizing and combining them into one data set. Pack 2 will have more weight than the other packs because it has the highest accuracy rate. The non-weighted set assumes that all features are equally important when they are not. Hence, we weigh the features to account for the importance of different features.

Weighted Trainset Confusion Matrix Weighted Trainset Confusion Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Train |  | PREDICTION | | | |
|  | **Test K = 5** | | **CL1** | **CL2** | **CL3** |
| TRUE | **CL1** | | 73.89% | 17.73% | 8.37% |
| **CL2** | | 17.05% | 77.84% | 5.11% |
| **CL3** | | 5.70% | 8.81% | 85.49% |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test |  | PREDICTION | | |
|  | **Test K = 5** | **CL1** | **CL2** | **CL3** |
| TRUE | **CL1** | 75.47% | 15.09% | 9.43% |
| **CL2** | 8.51% | 78.72% | 12.77% |
| **CL3** | 9.09% | 6.82% | 84.09% |

The global train set accuracy = 79.07%

The global test set accuracy = 79.42%

As we see, the global test accuracy is higher than our trainset, which is a good sign that there is no overfitting. There is an increase in accuracy in CL1 and CL2 when we move from trainset to test set, while there is a slight loss of accuracy in CL3 when we move from train set to test set.

Ordinary Distance KNN Test Set Weighted Distance KNN Test Set

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Weighted |  | PREDICTION | | |
| TEST | **Test K = 5** | **CL1** | **CL2** | **CL3** |
| TRUE | **CL1** | 75.47% | 15.09% | 9.43% |
| **CL2** | 8.51% | 78.72% | 12.77% |
| **CL3** | 9.09% | 6.82% | 84.09% |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ordinary |  | PREDICTION | | |
| TEST | **Test K = 5** | **CL1** | **CL2** | **CL3** |
| TRUE | **CL1** | 64.29% | 28.57% | 7.14% |
| **CL2** | 21.21% | 69.70% | 9.09% |
| **CL3** | 9.09% | 16.36% | 74.55% |

Ordinary KNN: Global test set accuracy = 69.51%

Weighted KNN: Global test set accuracy = 79.42%

The performance of the knn algorithm improved drastically after implementing the weighted distance knn. Our global performance increased by approximately 10%. Similarly, all three classes see an increase in accuracy by around 10%

.

*Appendix:*

Mean table:

A picture containing table, calendar

Description automatically generated

A picture containing table, calendar

Description automatically generated

Standard Deviation table:

A picture containing table

Description automatically generated

A picture containing table

Description automatically generated

Full script:

*attach*(BELL)  
*library*(dplyr) #Package for subseting data  
bell\_clean<- *select*(BELL,-*c*(2,3,6,7,8,9,10,11,12))#discard the 9 columns  
*View*(bell\_clean)  
#956x403  
  
*attach*(CHILLER)  
chiller\_clean<- *select*(CHILLER,-*c*(2,3,6,7,8,9,10,11,12))  
*View*(chiller\_clean)  
#952x403  
  
*attach*(FREESTYLE)  
freestyle\_clean<- *select*(FREESTYLE,-*c*(2,3,6,7,8,9,10,11,12))  
*View*(freestyle\_clean)  
#956x403  
  
#disccarding row containing missing numerical data  
BELL\_clean<- *na.omit*(bell\_clean)  
CHILLER\_clean<- *na.omit*(chiller\_clean)  
FREESTYLE\_clean<- *na.omit*(freestyle\_clean)  
  
#Defining three classes of images of normal characters  
#cl1 = all rows of BELL\_clean.csv file for which (strength = 0.4 and italic =0)  
#cl2 = all rows of CHILLER\_clean.csv file for which (strength = 0.4 and italic =0)  
#cl3 = all rows of FREESTYLE\_clean.csv file for which (strength = 0.4 and italic =0)  
  
BELL\_clean<-*data.frame*(BELL\_clean)#creating a data frame to add conditional statements to filter out non needed i features  
BELL\_clean$CL = *ifelse*((BELL\_clean$strength == 0.4 & BELL\_clean$italic == 0),"CL1","NA")  
BELL\_CLEAN = BELL\_clean[*which*(BELL\_clean$CL =="CL1"),] #labeling the new filter data as cl1  
*View*(BELL\_CLEAN)  
#404 columns  
#239 rows  
  
CHILLER\_clean<-*data.frame*(CHILLER\_clean)  
CHILLER\_clean$CL = *ifelse*((CHILLER\_clean$strength == 0.4 & CHILLER\_clean$italic == 0),"CL2","NA")  
CHILLER\_CLEAN = CHILLER\_clean[*which*(CHILLER\_clean$CL =="CL2"),]  
*View*(CHILLER\_CLEAN)  
#404 columns  
#238 rows  
  
FREESTYLE\_clean<-*data.frame*(FREESTYLE\_clean)  
FREESTYLE\_clean$CL = *ifelse*((FREESTYLE\_clean$strength == 0.4 & FREESTYLE\_clean$italic == 0),"CL3","NA")  
FREESTYLE\_CLEAN = FREESTYLE\_clean[*which*(FREESTYLE\_clean$CL =="CL3"),]  
*View*(FREESTYLE\_CLEAN)  
#404 columns  
#239 rows  
  
# Combine CL1, CL2, CL3 into DATA  
DATA<-*rbind*(BELL\_CLEAN,CHILLER\_CLEAN,FREESTYLE\_CLEAN)  
*View*(DATA)  
# Binded all 3 data sets to a full data set (DATA) which is the union of 3 classes (CL1, CL2, CL3)  
# where N = 716  
  
# Part 0  
# Compute mean  
DATAMEAN<-DATA %>% *summarize\_if*(is.numeric,mean)  
*mean*(DATA[,3]) #mean of this column is 0 which is ok  
# Compute standard deviation  
DATASD<-DATA %>% *summarize\_if*(is.numeric, sd)  
*var*(DATA[,3]) #sd is 0 which is not okay, which we need to standardize to have a comparable scale  
###standardizing to make a comparable scale  
*library*(standardize)  
SDATA<- DATA %>% *mutate\_if*(is.numeric, function (x) *as.vector*(*scale*(x))) # scaling by (xj-mj)/sd  
SDATA = SDATA[,-*c*(2,3)] #taking out numerical functions of strength and italics  
sDATA<- *data.matrix*(SDATA) #creating it into a data matrix for correlation matrix beforehand  
*View*(SDATA)  
### SDATA contains CL classes and font name, but not strength and italics.  
### confirming the standardization has properly worked by looking at mean and sd again of the SDATA  
*var*(SDATA[,3])#sd =1 which is good  
  
  
  
# Scale the data again for the  
sDATA1<-*scale*(sDATA[,-*c*(1,2,3,404)])# scaling and removing non-numerical values  
sDATA1  
#### sDATA1 is data set containing standardized features, but without non-numerical values.  
# correlation matrix  
*cor*(sDATA1)  
cor.df = *data.frame*(*cor*(DATA1)) #renaming to view actual full matrix  
*View*(cor.df)  
  
  
  
#packages needed to find the top 10 values  
*library*(dplyr)  
*library*(tidyr)  
#finding the top 10 highest values  
topvalues\_sdata<-*cor*(sDATA1) %>%  
 *as.data.frame*() %>%  
 *mutate*(var1 = *rownames*(.)) %>%  
 *gather*(var2, value, -var1) %>%  
 *arrange*(*desc*(value)) %>%  
 *group\_by*(value) %>%  
 *filter*(*row\_number*()==1)  
*View*(topvalues\_sdata)  
  
  
  
## loop to classify cl to SETROW columns into data set  
SDATA$SETROW<-NA  
  
for (i in 1:716){  
 if(SDATA$CL[i]=="CL1"){  
 SDATA$SETROW[i] = "SETROW1"  
}else if(SDATA$CL[i]=="CL2"){  
 SDATA$SETROW[i] = "SETROW2"  
}else{  
 SDATA$SETROW[i] = "SETROW3"  
}  
}  
#creating the 80% random train set interval by taking ONLY using setrow 1, we replicate this for the other setrow functions  
SETROW1 = SDATA[*which*(SDATA$SETROW =="SETROW1"),]  
n<-*nrow*(SETROW1[*which*(SETROW1$SETROW=="SETROW1"),])  
trainset<-*sample*(1:n, 0.8\*n)  
#  
trainsetcl1 <- SETROW1[trainset,]  
testsetcl1 <- SETROW1[-trainset,]  
  
SETROW2 = SDATA[*which*(SDATA$SETROW =="SETROW2"),]  
n<-*nrow*(SETROW2[*which*(SETROW2$SETROW=="SETROW2"),])  
trainset<-*sample*(1:n, 0.8\*n)  
trainsetcl2 <- SETROW2[trainset,]  
testsetcl2 <- SETROW2[-trainset,]  
  
SETROW3 = SDATA[*which*(SDATA$SETROW =="SETROW3"),]  
n<-*nrow*(SETROW3[*which*(SETROW3$SETROW=="SETROW3"),])  
trainset<-*sample*(1:n, 0.8\*n)  
trainsetcl3 <- SETROW3[trainset,]  
testsetcl3 <- SETROW3[-trainset,]  
  
#combining the sets to full trainset and testset  
  
TRAIN\_SET<-*rbind*(trainsetcl1,trainsetcl2,trainsetcl3)  
TEST\_SET<- *rbind*(testsetcl1,testsetcl2,testsetcl3)  
  
#train and test labels  
*library*(class)  
SDATA\_no <- SDATA[,-*c*(1,402,403)]  
SDATA\_label <- SDATA[,"CL"]  
TRAIN\_no <- TRAIN\_SET[,-*c*(1,402,403)]  
TRAIN\_label <- TRAIN\_SET[, "CL"]  
TEST\_no <- TEST\_SET[,-*c*(1,402,403)]  
TEST\_label <- TEST\_SET[,"CL"]

# Compute the percentage of correct classification  
*RNGkind*(sample.kind = "Rounding")  
*set.seed*(1)  
knn.predtrain12 <- *knn*(train=TRAIN\_no,  
 test=TRAIN\_no,  
 cl = TRAIN\_label,  
 k=12)  
  
*RNGkind*(sample.kind = "Rounding")  
*set.seed*(1)  
knn.predtest12 <- *knn*(train=TRAIN\_no,  
 test=TEST\_no,  
 cl = TRAIN\_label,  
 k=12)  
  
*mean*(knn.predtrain12!= TRAIN\_label) #[1] 0.2814685 pretty bad  
*mean*(knn.predtest12 != TEST\_label) #[1] 0.3194444 very high which makes sense because its the test set  
  
#confusion matrix  
*table*(*data.frame*(knn.predtrain,TRAIN\_label))  
*table*(*data.frame*(knn.predtest, TEST\_label))  
  
#1.2  
#fit the model on the training set finding the optimized value of k for test set  
#running a loop  
*set.seed*(1)  
K.set = *c*(5,10,15,20,25,30,40,50,100)  
knn.test.accuracy <- *numeric*(*length*(K.set))  
  
for (j in 1:*length*(K.set)){  
 knn.pred <- *knn*(train=TRAIN\_no,  
 test=TEST\_no,  
 cl=TRAIN\_label,  
 k=K.set[j])  
 knn.test.accuracy[j] <- *mean*(knn.pred == TEST\_label)  
}  
  
####finding percent accuracy for each value of 5,10...100  
*set.seed*(1)  
i=1  
k.optm=1  
for (i in *seq*(5, 100, by = 5)){  
 knn.mod<- *knn*(train = TRAIN\_no, test = TEST\_no, cl = TRAIN\_label, k= i)  
 k.optm[i]<- 100 \* *sum*(TEST\_label == knn.mod)/ *NROW*(TEST\_label)  
 k=i  
 *cat*(k,"=", k.optm[i],'\n')  
}  
  
### finding accuracy for train which will be higher  
*set.seed*(1)  
K.set = *c*(5,10,15,20,25,30,40,50,100)  
knn.train.accuracy <- *numeric*(*length*(K.set))  
  
for (j in 1:*length*(K.set)){  
 knn.pred <- *knn*(train=TRAIN\_no,  
 test=TRAIN\_no,  
 cl=TRAIN\_label,  
 k=K.set[j])  
 knn.train.accuracy[j] <- *mean*(knn.pred == TRAIN\_label)  
}  
###plotting the figure with each other.  
##red = test set  
## blue = trainset  
  
*plot*(K.set, knn.train.accuracy, type="o", col="blue", pch="o", lty=1 )  
*points*(K.set, knn.test.accuracy, col="red", pch="\*")  
*lines*(K.set, knn.test.accuracy, col="red",lty=2)  
  
#### based on the figure we can do 5:20 range to explore more k values  
K.set = *c*(5:20)  
knn.test.accuracy <- *numeric*(*length*(K.set))  
*set.seed*(1)  
for (j in 1:*length*(K.set)){  
 knn.pred <- *knn*(train=TRAIN\_no,  
 test=TEST\_no,  
 cl=TRAIN\_label,  
 k=K.set[j])  
 knn.test.accuracy[j] <- *mean*(knn.pred == TEST\_label)  
}  
# Find best k  
*max*(knn.test.accuracy)  
K.set[*which.max*(knn.test.accuracy)]  
##based on the information provided, we use K=5 as k best  
  
##Applying the "best" k value to the both train and test set.  
*set.seed*(1)  
knn.predtrainbest<- *knn*(train=TRAIN\_no,  
 test=TRAIN\_no,  
 cl = TRAIN\_label,  
 k=5)  
*set.seed*(1)  
knn.predtestbest <- *knn*(train=TRAIN\_no,  
 test=TEST\_no,  
 cl = TRAIN\_label,  
 k=5)  
##displaying the percent error values  
*mean*(knn.predtrainbest == TRAIN\_label)  
*mean*(knn.predtestbest == TEST\_label)  
  
##displaying confusion matrix of cl   
trainmt<-*table*(*data.frame*(knn.predtrainbest,TRAIN\_label))  
testtt<-*table*(*data.frame*(knn.predtestbest, TEST\_label))  
  
#####finding confidence intervals of confusion matrix  
*library*(DescTools)  
*Conf*(trainmt)  
#yielding 95% CI : (0.7975, 0.8589)  
*Conf*(testtt)  
#yielding 95% CI : (0.6734, 0.8136)  
  
###1.6  
###PACK 1 L: 0-9 and M: 0-9 making a 100 attributes  
PACK1<-SDATA[,*c*(2:11,22:31,42:51,62:71,82:91,102:111,122:131,142:151,162:171,182:191,402,403)]  
PACK2<-SDATA[,*c*(12:21,32:41,52:61,72:81,92:101,112:121,132:141,152:161,172:181,192:201,402,403)]  
PACK3<-SDATA[,*c*(212:221,232:241,252:261,272:281,292:301,312:321,332:341,352:361,372:381,392:401,402,403)]  
PACK4<-SDATA[,*c*(202:211,222:231,242:251,262:271,282:291,302:311,322:331,342:351,362:371,382:391,402,403)]  
  
##dividing set of pack 1 of .8 of train .2 test  
packcl1 = PACK1[*which*(PACK1$SETROW=="SETROW1"),]  
n<-*nrow*(packcl1)  
PACKCL11<-*sample*(1:n, 0.8\*n)  
  
PACKCL1\_train1 <- packcl1[PACKCL11,]  
PACKCL1\_test1 <- packcl1[-PACKCL11,]  
  
#replicating for cl2,  
packcl2 = PACK1[*which*(PACK1$SETROW=="SETROW2"),]  
n<-*nrow*(packcl2)  
PACKCL21<-*sample*(1:n, 0.8\*n)  
  
PACKCL2\_train1 <- packcl2[PACKCL21,]  
PACKCL2\_test1 <- packcl2[-PACKCL21,]  
  
### replicating for cl3  
packcl3 = PACK1[*which*(PACK1$SETROW=="SETROW3"),]  
n<-*nrow*(packcl3)  
PACKCL31<-*sample*(1:n, 0.8\*n)  
  
PACKCL3\_train1 <- packcl3[PACKCL31,]  
PACKCL3\_test1 <- packcl3[-PACKCL31,]  
####UNIONIZING PACK1 CLs  
PACK1\_TRAINALL<- *rbind*(PACKCL1\_train1,PACKCL2\_train1,PACKCL3\_train1)  
PACK1\_TESTALL<- *rbind*(PACKCL1\_test1,PACKCL2\_test1,PACKCL3\_test1)  
  
######PACK 2 #######  
##dividing set of pack 1 of .8 of train .2 test  
  
packcl1p2 = PACK2[*which*(PACK2$SETROW=="SETROW1"),]  
n<-*nrow*(packcl1p2)  
PACKCL1p2<-*sample*(1:n, 0.8\*n)  
  
PACKCL1\_train2 <- packcl1p2[PACKCL1p2,]  
PACKCL1\_test2 <- packcl1p2[-PACKCL1p2,]  
  
#pack2 cl2  
packcl2p2 = PACK2[*which*(PACK2$SETROW=="SETROW2"),]  
n<-*nrow*(packcl2p2)  
PACKCL2p2<-*sample*(1:n, 0.8\*n)  
  
PACKCL2\_train2 <- packcl2p2[PACKCL2p2,]  
PACKCL2\_test2 <- packcl2p2[-PACKCL2p2,]  
  
#pack 2 cl3  
packcl3p2 = PACK2[*which*(PACK2$SETROW=="SETROW1"),]  
n<-*nrow*(packcl3p2)  
PACKCL3p2<-*sample*(1:n, 0.8\*n)  
  
PACKCL3\_train2 <- packcl3p2[PACKCL3p2,]  
PACKCL3\_test2 <- packcl3p2[-PACKCL3p2,]  
  
PACK2\_TRAINALL<- *rbind*(PACKCL1\_train2,PACKCL2\_train2,PACKCL3\_train2)  
PACK2\_TESTALL<- *rbind*(PACKCL1\_test2,PACKCL2\_test2,PACKCL3\_test2)  
  
######PACK 3  
##CL1  
packcl1p3 = PACK3[*which*(PACK3$SETROW=="SETROW1"),]  
n<-*nrow*(packcl1p3)  
PACKCL1p3<-*sample*(1:n, 0.8\*n)  
  
PACKCL1\_train3 <- packcl1p3[PACKCL1p3,]  
PACKCL1\_test3 <- packcl1p3[-PACKCL1p3,]  
#####CL2  
  
packcl2p3 = PACK3[*which*(PACK3$SETROW=="SETROW2"),]  
n<-*nrow*(packcl2p3)  
PACKCL2p3<-*sample*(1:n, 0.8\*n)  
  
PACKCL2\_train3 <- packcl2p3[PACKCL2p3,]  
PACKCL2\_test3 <- packcl2p3[-PACKCL2p3,]  
  
###CL3  
packcl3p3 = PACK3[*which*(PACK3$SETROW=="SETROW3"),]  
n<-*nrow*(packcl3p3)  
PACKCL3p3<-*sample*(1:n, 0.8\*n)  
  
PACKCL3\_train3 <- packcl3p3[PACKCL3p3,]  
PACKCL3\_test3 <- packcl3p3[-PACKCL3p3,]  
  
PACK3\_TRAINALL<- *rbind*(PACKCL1\_train3,PACKCL2\_train3,PACKCL3\_train3)  
PACK3\_TESTALL<- *rbind*(PACKCL1\_test3,PACKCL2\_test3,PACKCL3\_test3)  
  
###PACK 4  
#CL1  
packcl1p4 = PACK4[*which*(PACK4$SETROW=="SETROW1"),]  
n<-*nrow*(packcl1p4)  
PACKCL1p4<-*sample*(1:n, 0.8\*n)  
  
PACKCL1\_train4 <- packcl1p4[PACKCL1p4,]  
PACKCL1\_test4 <- packcl1p4[-PACKCL1p4,]  
  
#CL2  
packcl2p4 = PACK4[*which*(PACK4$SETROW=="SETROW2"),]  
n<-*nrow*(packcl2p4)  
PACKCL2p4<-*sample*(1:n, 0.8\*n)  
  
PACKCL2\_train4 <- packcl2p4[PACKCL2p4,]  
PACKCL2\_test4 <- packcl2p4[-PACKCL2p4,]  
  
#CL3  
packcl3p4 = PACK4[*which*(PACK4$SETROW=="SETROW3"),]  
n<-*nrow*(packcl3p4)  
PACKCL3p4<-*sample*(1:n, 0.8\*n)  
  
PACKCL3\_train4 <- packcl3p4[PACKCL3p4,]  
PACKCL3\_test4 <- packcl3p4[-PACKCL3p4,]  
  
PACK4\_TRAINALL<- *rbind*(PACKCL1\_train4,PACKCL2\_train4,PACKCL3\_train4)  
PACK4\_TESTALL<- *rbind*(PACKCL1\_test4,PACKCL2\_test4,PACKCL3\_test4)  
  
#################labeling test entry for knn ##################  
  
PACK1\_TRAINALL\_no<- PACK1\_TRAINALL[,-*c*(101,102)]  
PACK1\_TRAINALL\_LABEL <- PACK1\_TRAINALL[,"CL"]  
PACK2\_TRAINALL\_no<- PACK2\_TRAINALL[,-*c*(101,102)]  
PACK2\_TRAINALL\_LABEL <- PACK2\_TRAINALL[,"CL"]  
PACK3\_TRAINALL\_no<- PACK3\_TRAINALL[,-*c*(101,102)]  
PACK3\_TRAINALL\_LABEL <- PACK3\_TRAINALL[,"CL"]  
PACK4\_TRAINALL\_no<- PACK4\_TRAINALL[,-*c*(101,102)]  
PACK4\_TRAINALL\_LABEL <- PACK4\_TRAINALL[,"CL"]  
  
PACK1\_TESTALL\_no<- PACK1\_TESTALL[,-*c*(101,102)]  
PACK1\_TESTALL\_LABEL <- PACK1\_TESTALL[,"CL"]  
PACK2\_TESTALL\_no<- PACK2\_TESTALL[,-*c*(101,102)]  
PACK2\_TESTALL\_LABEL <- PACK2\_TESTALL[,"CL"]  
PACK3\_TESTALL\_no<- PACK3\_TESTALL[,-*c*(101,102)]  
PACK3\_TESTALL\_LABEL <- PACK3\_TESTALL[,"CL"]  
PACK4\_TESTALL\_no<- PACK4\_TESTALL[,-*c*(101,102)]  
PACK4\_TESTALL\_LABEL <- PACK4\_TESTALL[,"CL"]  
  
# Apply KNN using K = 5 to all four pack testssets  
*set.seed*(1)  
knn.predPACK1 <- *knn*(train=PACK1\_TRAINALL\_no,  
 test=PACK1\_TESTALL\_no,  
 cl = PACK1\_TRAINALL\_LABEL,  
 k=5)  
  
  
*set.seed*(1)  
knn.predPACK2 <- *knn*(train=PACK2\_TRAINALL\_no,  
 test=PACK2\_TESTALL\_no,  
 cl = PACK2\_TRAINALL\_LABEL,  
 k=5)  
  
  
  
*set.seed*(1)  
knn.predPACK3 <- *knn*(train=PACK3\_TRAINALL\_no,  
 test=PACK3\_TESTALL\_no,  
 cl = PACK3\_TRAINALL\_LABEL,  
 k=5)  
  
  
  
*set.seed*(1)  
knn.predPACK4 <- *knn*(train=PACK4\_TRAINALL\_no,  
 test=PACK4\_TESTALL\_no,  
 cl = PACK4\_TRAINALL\_LABEL,  
 k=5)  
  
  
  
# Find accuracy and set it to weight  
w1 <- *mean*(knn.predPACK1 == PACK1\_TESTALL\_LABEL)  
w2 <- *mean*(knn.predPACK2 == PACK2\_TESTALL\_LABEL)  
w3 <- *mean*(knn.predPACK3 == PACK3\_TESTALL\_LABEL)  
w4 <- *mean*(knn.predPACK4 == PACK4\_TESTALL\_LABEL)  
#displaying values of accuracy   
  
w1 #0.5486111  
w2 #0.7569444  
w3 #0.6319444  
w4 #0.6666667  
### we can see pack2 had the highest accuracy here  
  
# Multiply weights to each pack  
wpack1<-PACK1\_no\*w1  
wpack2<-PACK2\_no\*w2  
wpack3<-PACK3\_no\*w3  
wpack4<-PACK4\_no\*w4  
  
#binding the weighted packs  
wpackfull<- *cbind*(wpack1,wpack2,wpack3,wpack4,PACK1[,101])  
*View*(wpackfull)  
#### normalizing the full weight packs   
Swpackfull<- wpackfull %>% *mutate\_if*(is.numeric, function (x) *as.vector*(*scale*(x))

#LABELS

waptrainset\_no <- waptrainset[,-401]  
waptrainset\_label <- waptrainset[,401]  
  
waptestset\_no <- waptestset[,-401]  
waptestset\_label <- waptestset[,401]

# Global Performance for both train and test set with weighted values where knn= 5  
*set.seed*(1)  
  
knn.predwtrain <- *knn*(train=waptrainset\_no,  
 test=waptrainset\_no,  
 cl = waptrainset\_label,  
 k=5)  
  
full.predwtrain<- *mean*(knn.predwtrain == waptrainset\_label)  
full.predwtrain #0.78 accuracy yay  
  
##testset  
knn.predwtest <- *knn*(train=waptrainset\_no,  
 test=waptestset\_no,  
 cl = waptrainset\_label,  
 k=5)  
full.predwtest<- *mean*(knn.predwtest == waptestset\_label)  
full.predwtest  
  
#confusion matrix  
*table*(*data.frame*(knn.predwtrain,waptrainset\_label))  
*table*(*data.frame*(knn.predwtest, waptestset\_label))