STA 141A Spring 2018

Final Project

Jiani Wang, Kevin Lee, Min Woo Kim, Zihan Mo

Q1. See appendix

Q2. See appendix

Q3.

Airplane

Automobile

Bird

Cat

Deer

Dog

Frog

Horse

Ship

Truck

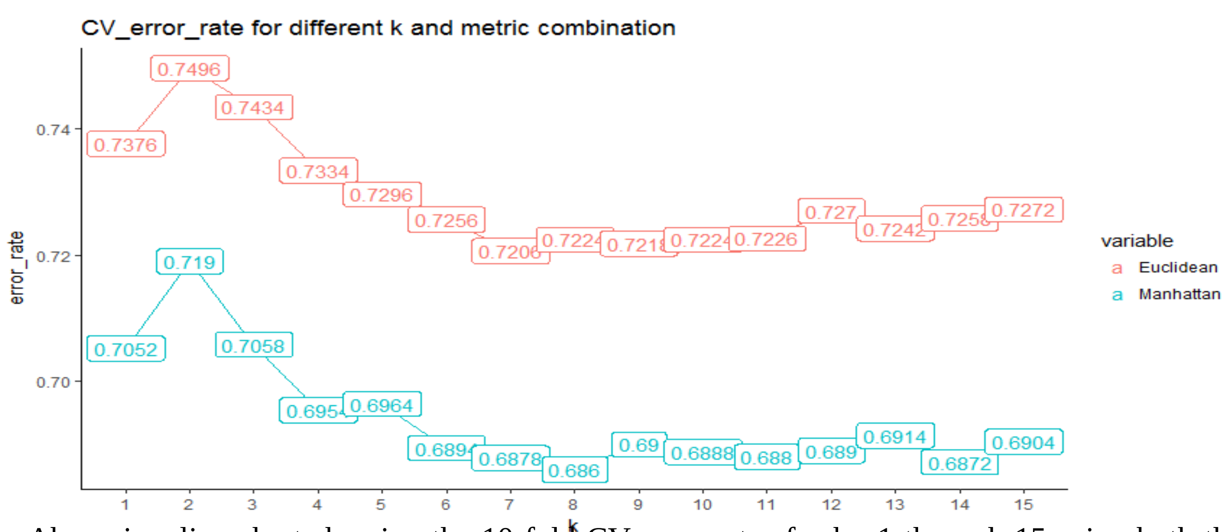
To determine if a specific pixel is informative, we need to check if its color (value) change a lot. Therefore, we check the standard deviation for all the pixels through all the images. The 5 most useful pixels are 2049 (81.69) in blue channel, 2081(80.91), 2080(80.86), 2050(80.69), and 2079(80.45) in green channel. The 5 least useful pixels are 1424(57.46), 1396(57.44), 1392(57.38), 1395(57.38), and 1391(57.28).

Most of the subjects in the images are in the center, so looking at the pixels at the center of the data would be most effective for classification.

Q4. See appendix

Q5. In order to increase efficiency, our cv_error_knn() function takes in distance matrix to predict labels for the prediction point(s). Therefore the distance matrix is calculated once beforehand outside the function, so we don't need to recalculate the distance matrices for each k. Furthermore, we maximized the use of vectorized operations, which is faster than using for-loops. Vectorized operations were used to calculate which folds each index of the training data would go into (using rep_len), dividing the training dataset into the appropriate training and testing bins, checking if the predicted label was correct using the '!=' operator on the vectors, and calculating the number of incorrect labels by using the sum() function on a vector of Booleans. We also initialized the folderrors list before the for-loop which calculates and stores every error in the folderrors list.

Q6.



Above is a line chart showing the 10-fold CV error rates for k= 1 through 15 using both the Euclidean and Manhattan distance metrics. Based on this plot, the 3 best k with Euclidean distance metric are 7, 9, and 8. The 3 best k with Manhattan distance metric are 8, 14, and 7. Obviously, Manhattan metric has lower error rate than the Euclidean metric for all corresponding k. Since the error rate doesn't change dramatically after k equals to 6, no additional values of k would be useful.

When we encounter a tie situation, we assign the label randomly from the classifications with the highest count. Therefore, the best k could vary, but best ks are always greater than 5. Even though the best k could vary, the trend and error rate do not change significantly.

Q7.

Euclidean with k = 7

pre_labels \ act_labels	0	1	2	3	4	5	6	7	8	9
0	216	43	47	38	28	24	18	43	93	52
1	2	38	2	0	2	1	0	3	3	13
2	71	45	191	104	109	84	122	97	39	50
3	9	27	20	61	20	45	26	19	10	16
4	33	121	156	127	250	145	156	167	42	68
5	4	12	7	50	8	91	21	15	14	13
6	26	58	36	81	45	68	145	51	7	42
7	9	13	5	12	15	13	5	62	7	16
8	129	122	36	26	22	26	6	33	274	170
9	1	21	0	1	1	3	1	10	11	60

Euclidean with k = 8

pre_labels \ act_labels	0	1	2	3	4	5	6	7	8	9
0	224	43	48	30	28	33	12	39	101	54
1	0	31	1	0	1	1	0	1	3	14
2	67	48	191	112	122	93	132	90	31	53
3	5	25	19	64	17	48	21	21	9	16
4	36	127	157	127	247	140	154	169	44	69
5	2	19	8	47	5	88	19	19	13	10
6	30	48	35	75	47	62	151	48	9	37
7	4	8	6	12	13	7	5	62	6	18
8	131	128	34	32	20	25	5	41	276	170
9	1	23	1	1	0	3	1	10	8	59

Euclidean with k = 9

pre_labels \ act_labels	0	1	2	3	4	5	6	7	8	9
0	214	39	52	34	27	29	11	44	96	54
1	0	31	1	0	1	1	2	1	5	16
2	71	61	197	107	115	94	128	102	28	53
3	7	17	17	58	9	46	22	13	8	13
4	35	129	150	133	266	144	159	173	42	69
5	4	19	8	47	6	92	15	17	12	12
6	28	45	36	72	37	57	148	43	6	37
7	5	12	5	14	14	11	9	61	4	17
8	135	127	33	33	24	24	5	38	290	170
9	1	20	1	2	1	2	1	8	9	59

Manhattan with k =7

pre_labels \ act_labels	0	1	2	3	4	5	6	7	8	9
0	232	46	53	37	32	37	18	42	99	63
1	2	71	1	5	1	2	6	2	5	29
2	73	52	198	95	115	91	137	79	28	41
3	3	29	22	83	12	66	26	19	3	25
4	22	92	138	117	248	108	126	150	33	49
5	3	15	5	49	6	102	15	17	12	14
6	23	53	43	68	46	55	156	41	12	26
7	7	16	10	16	22	10	7	95	6	27
8	131	100	25	27	15	26	8	39	290	122
9	4	26	5	3	3	3	1	16	12	104

Manhattan with k = 8

act_labels

pre_labels	0	1	2	3	4	5	6	7	8	9
0	232	50	54	34	36	31	15	41	100	57
1	1	72	2	5	1	2	3	2	5	32
2	71	55	199	92	120	87	133	93	27	36
3	5	18	24	69	15	52	22	20	4	21
4	22	104	131	126	245	118	135	152	31	52
5	2	19	8	47	3	102	18	14	12	8
6	28	51	36	77	39	59	160	34	9	36
7	5	11	8	20	19	13	5	89	8	33
8	131	91	32	26	18	31	7	39	294	131
9	3	29	6	4	4	5	2	16	10	94

Manhattan with k =14

act_labels

pre_labels	0	1	2	3	4	5	6	7	8	9
0	243	35	57	36	35	32	16	48	96	50
1	2	48	0	1	2	4	4	0	3	23
2	56	61	202	101	122	100	130	82	26	37
3	6	19	17	64	7	42	20	13	4	18
4	27	106	128	126	248	127	144	172	37	49
5	2	17	7	41	8	96	12	11	10	8
6	24	54	40	80	42	55	161	34	9	38
7	7	10	9	15	14	10	6	91	6	28
8	129	112	38	28	20	28	6	32	298	140
9	4	38	2	8	2	6	1	17	11	109

Based on the confusion matrices above, there are more predicted points assigned to labels 0, 2, 4, 6, and 8 (which correspond to airplane, bird, deer, frog, and ship) for all the combinations of k and distance metrics. The distribution of misclassified labels and classified labels for different k using the same distance metric is very similar, so we do not need to choose a different k-value. It's still evident that Manhattan metric has higher accuracy than the Euclidean metric with corresponding k. Therefore, Manhattan combination is still more preferable than Euclidean metric.

Q8.

Manhattan with k =7

act_labels

pre_labels	0	1	2	3	4	5	6	7	8	9
0	232	46	53	37	32	37	18	42	99	63
1	2	71	1	5	1	2	6	2	5	29
2	73	52	198	95	115	91	137	79	28	41
3	3	29	22	83	12	66	26	19	3	25
4	22	92	138	117	248	108	126	150	33	49
5	3	15	5	49	6	102	15	17	12	14
6	23	53	43	68	46	55	156	41	12	26
7	7	16	10	16	22	10	7	95	6	27
8	131	100	25	27	15	26	8	39	290	122
9	4	26	5	3	3	3	1	16	12	104

Manhattan with k = 8

act_labels

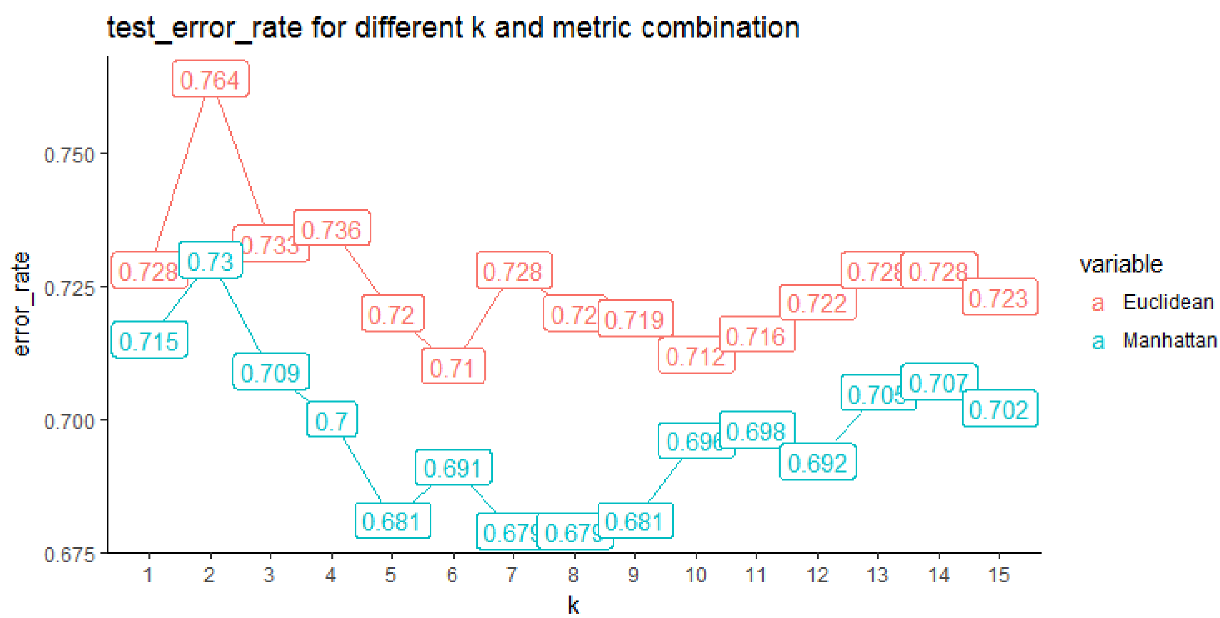
pre_labels	0	1	2	3	4	5	6	7	8	9
0	232	50	54	34	36	31	15	41	100	57
1	1	72	2	5	1	2	3	2	5	32
2	71	55	199	92	120	87	133	93	27	36
3	5	18	24	69	15	52	22	20	4	21
4	22	104	131	126	245	118	135	152	31	52
5	2	19	8	47	3	102	18	14	12	8
6	28	51	36	77	39	59	160	34	9	36
7	5	11	8	20	19	13	5	89	8	33
8	131	91	32	26	18	31	7	39	294	131
9	3	29	6	4	4	5	2	16	10	94

Manhattan with k =14

pre_labels \ act_labels	0	1	2	3	4	5	6	7	8	9
0	243	35	57	36	35	32	16	48	96	50
1	2	48	0	1	2	4	4	0	3	23
2	56	61	202	101	122	100	130	82	26	37
3	6	19	17	64	7	42	20	13	4	18
4	27	106	128	126	248	127	144	172	37	49
5	2	17	7	41	8	96	12	11	10	8
6	24	54	40	80	42	55	161	34	9	38
7	7	10	9	15	14	10	6	91	6	28
8	129	112	38	28	20	28	6	32	298	140
9	4	38	2	8	2	6	1	17	11	109

As mentioned above, the classifier is in favor of assigning prediction point to labels 0, 2, 4, 6, and 8 (which correspond to airplane, bird, deer, frog, and ship). Furthermore, these labels also have more misclassified labels. Especially for labels 2 and 4, there are a large amount of misclassified labels. Therefore, to increase the accuracy, we should modify our classifier focus on labelling 2 and 4 (which corresponds to bird and deer).

Q9.



From the plot above, we could find that we can get lowest error rate when k is 6, 10, and 11 with the Euclidean distance metric. Additionally, we get the best error rate when Manhattan distance metric is used with k = 7, 8, and 9. Like the problem 6, the Manhattan metric has lower error rate than the Euclidean metric for all corresponding k. Also similar is that the error rate remains relatively steady for k-values over 6, so we do not need to test more k-values.

When we compare the results to the 10-fold CV error rates, we found that the plot looks similar which means that the trend and error rate do not change a lot. However, the error rates in the 10-fold CV plot were more stable when k is larger than 6.

Q10.

Kevin Lee and Zihan Mo focused on understanding the logic and coding.

Min Woo Kim, Jiani Wang did some research and analyzed answers for the report.

References:

https://stats.stackexchange.com/questions/61090/how-to-split-a-data-set-to-do-10-fold-cross-validation

Patrick’s TA Office Hour

Professor Gupta’s Office Hour and Lecture

Jiahui’s Discussion Section

STA141A Piazza.com

Appendix

```
library(grid)
library(ggplot2)
library(reshape)
#Q1
# function to read in binary image files from a directory file_path and output a RDS file at
out_file
load_training_images<-function(file_path,out_file){
  # take input directory path and add each training file to path, store as vector
  files<-file.path(file_path,c("data_batch_1.bin","data_batch_2.bin","data_batch_3.bin",
                               "data_batch_4.bin","data_batch_5.bin"))
  # read in each training binary file and store as list for each file
  d<-lapply(files,function(x)readBin(con=x,what = raw(),n=30730000))
  # unlist the training data
  d<-unlist(d)
  # generate indices of label for each image
  labelindex<-seq(1,153650000,3073)
  # get labels for all the training images
  labels<-as.integer(d[labelindex])
  # read in pixel data for each image and store as matrix
  bins<-matrix(as.integer(d[-labelindex]),50000,3072,byrow=TRUE)
  # bind the label and pixel data for each image
  images<-cbind(labels,bins)
  # save images matrix to rds file
  saveRDS(images,file=out_file)
}

# read in training image data from bin files
load_training_images("D:/R/FP",out_file = "D:/R/FP/training.rds")

# function to read in binary image file from file-path and output a RDS file at out_file
load_testing_image<-function(file_path,out_file){
  # read in testing binary file
  file<-readBin(con=file_path,what=raw(),n=30730000)
  # generate indices of label for each image
  labelindex<-seq(1,30730000,3073)
  # get labels for all the training images
  label<-as.integer(file[labelindex])
  # read in pixel data for each image and store as matrix
  bin<-matrix(as.integer(file[-labelindex]),10000,3072,byrow=TRUE)
  # bind the label and pixel data for each image
  image<-cbind(label,bin)
  # sace images matrix to rds file
  saveRDS(image,out_file)
}

# read in testing image data from bin file
load_testing_image(file_path = "D:/R/FP/test_batch.bin",out_file = "D:/R/FP/test.rds")

# scale down the training and testing datasets (code thanks to Patrick)
training<-readRDS("training.rds")
testing<-readRDS("test.rds")
data_rescale<-
function(labels,k=500)sort(as.vector(sapply(unique(labels),function(i)which(labels==i))[1:k,]))
train<-training[data_rescale(training[,1],k=500),]
test<-testing[data_rescale(testing[,1],k=100),]

#Q2
# read in label metadata file
class<-read.table("batches.meta.txt")

view_images<-function(images,ref,inx){
  r <- matrix(images[inx,c(2:1025)], ncol=32,byrow = T)
  g <- matrix(images[inx,c(1026:2049)], ncol=32,byrow = T)
  b <- matrix(images[inx,c(2050:3073)], ncol=32,byrow = T)
  col<-rgb(r,g,b,maxColorValue = 255)
  dim(col) <- dim(r)
  grid.raster(col,interpolate = F)
  label=as.character(ref[images[inx,1]+1,])
```

```
  print(label)
}

view_images<-function(images,ref,inx){
  # create 32x32 pixel matrices for each color channel
  r <- matrix(images[inx,c(2:1025)], ncol=32,byrow = T)
  g <- matrix(images[inx,c(1026:2049)], ncol=32,byrow = T)
  b <- matrix(images[inx,c(2050:3073)], ncol=32,byrow = T)
  # combine colors together and normalize the pixel intensities
  col<-rgb(r,g,b,maxColorValue = 255)
  dim(col) <- dim(r)
  # Now display the image
  grid.raster(col,interpolate = F)
  # map label integer to classification based on the reference
  label=as.character(ref[images[inx,1]+1,])
  # print classification
  print(label)
}

#Q3
# subset training pixel data (do not include labels)
train_pixel<-train[,-1]
# get standard deviation of pixels
sdpixel<-apply(train_pixel,2,sd)
# get 5 highest standard deviation pixels
head(sort(sdpixel,decreasing = T),5)
head(order(sdpixel,decreasing = T),5)
# get 5 lowest standard deviation pixels
tail(sort(sdpixel,decreasing = T),5)
tail(order(sdpixel,decreasing = T),5)

#Q4
# calculate Euclidean distance matrix or training images
eucdmatrix<-dist(train[,-1],upper=T,diag=T)
eucdmatrix<-as.matrix(eucdmatrix)
# save Euclidean distance matrix to rdS file
saveRDS(eucdmatrix,"D:/R/FP/eucdmatrix.rds")
# calculate manhattan distance matrix of training images
mandmatrix<-dist(train[,-1],upper=T,diag=T,method="manhattan")
mandmatrix<-as.matrix(mandmatrix)
# save Manhattan distance matrix to file
saveRDS(mandmatrix,"D:/R/FP/mandmatrix.rds")

# read in euclidean and manhattan distance matrices for training images
eucdmatrix<-readRDS('eucdmatrix.rds')
mandmatrix<-readRDS('mandmatrix.rds')

# function to randomly assign tie label, solving tie problem, x is vector of most likely
classification(s)
vote<-function(x){
  # randomly pick a label
  label<-as.integer(sample(names(which(x==max(x))),1,replace = T))
  return(label)
}

knn_predict<-function(predict,train,Dmatrix,k){
  # initialize predicted labels vector
  labels<-c(nrow(predict))
  # for all the prediction points
  for(i in 1:nrow(predict)){
    # get indices of k nearest neighbors
    nearest_k<-head(order(Dmatrix[i,]),n=k)
    # get the labels of the k nearest neighbors
    label_k<-train[c(nearest_k),1]
    # tabulate the labels of the k nearest neighbors
    counts<-table(label_k)
    # get and store the most likely label
    labels[i]<-vote(counts)
  }
  return(labels)
}
```

```
#Q5
cv_error_knn<-function(train,Dmatrix,k){
  # vector to assign training data into 10 same size folds
  folds <- rep_len(1:10,nrow(train))
  # initialize vector containing error rate for each fold
  folderrors<-numeric()
  # for each fold
  for(i in 1:10) {
    # get indices of test points
    test_inx<-which(folds==i)
    # get test points
    testing<-train[test_inx,]
    # get training points
    training<-train[-test_inx,]
    # subset distance matrix to only look at current testing points
    ttDmatrix<-Dmatrix[test_inx,-test_inx]
    # predict the labels for test points
    pre_labels<-knn_predict(testing,training,ttDmatrix,k)
    # subset the actual labels for test points
    act_labels<-testing[,1]
    # create boolean vector indicating incorrect labels
    incorrectlabels<-pre_labels!=act_labels
    # calculate error rate for current fold and store it in the vector
    folderrors[i]<-sum(incorrectlabels)/nrow(testing)
  }
  #return the error mean
  return(mean(folderrors))
}
cv_error_knn(train,eucdmatrix,7)

#Q6
#metrics and k(1:20) combination
euc_20_error<-numeric()
man_20_error<-numeric()
# for k = 1 through 20
for(k in 1:20){
  # calculate the error rate for 10 fold cross validation for euclidian and manhattan metrics
  euc_20_error[k]<-cv_error_knn(train,eucdmatrix,k)
  man_20_error[k]<-cv_error_knn(train,mandmatrix,k)
}
#metrics and k(1:15) combination
euc_15_error<-euc_20_error[1:15]
man_15_error<-man_20_error[1:15]
#data cleaning
# generate sequence of numbers 1 through 15 for each k-value
k<-seq(1,15,1)
# bind the error rates for euclidean and manhattan distance matrices together with the k value
error_rate<-data.frame(cbind(euc_15_error,man_15_error,k))
# give column names to data frame
colnames(error_rate)<-c("Euclidean","Manhattan","k")
error_rate<-melt(error_rate,id="k")
levels(error_rate$variable)<-c('Euclidean','Manhattan')
#visualization for the error rate for different combinations if k and distance metrics
ggplot(data=error_rate,aes(x=k,y=value,col=variable,label=value))+
  geom_point()+geom_line()+
  scale_x_continuous(breaks = c(seq(1,15,1)))+
  geom_label()+
  ylab("error_rate")+ggtitle("CV_error_rate for different k and metric combination")+
  theme_classic()
#check if other k is useful
best3_euc<-head(order(euc_20_error),3)
head(order(euc_15_error),4)
best3_man<-head(order(man_20_error),3)
head(order(man_15_error),3)
cv_error_knn(train,eucdmatrix,100)
cv_error_knn(train,eucdmatrix,90)

#Q7
#10 folds confusion matrix
cv_matrix_knn<-function(train,Dmatrix,k){
```

```
  # vector to assign training data into 10 same size folds
  folds <- rep_len(1:10,nrow(train))
  #vectorization first
  folderrors<-numeric()
  pre_labels<-c()
  act_labels<-c()
  # for each fold
  for(i in 1:10) {
    # get indices of test points
    test_inx<-which(folds==i)
    # get test points
    testing<-train[test_inx,]
    # get training points
    training<-train[-test_inx,]
    # subset distance matrix to only look at current testing points
    ttDmatrix<-Dmatrix[test_inx,-test_inx]
    # get prediction labels for test points
    pre_labels<-c(pre_labels,knn_predict(testing,training,ttDmatrix,k))
    # subset the actual labels for test points
    act_labels<-c(act_labels,testing[,1])
  }
  # return confusion matrix
  return(table(pre_labels,act_labels))
}
#Euclidean top 3 k combination confusion matrix
euc_con_7<-cv_matrix_knn(train,eucdmatrix,7)
euc_con_8<-cv_matrix_knn(train,eucdmatrix,8)
euc_con_9<-cv_matrix_knn(train,eucdmatrix,9)
#Manhattan top 3 k combination confusion matrix
man_con_7<-cv_matrix_knn(train,mandmatrix,7)
man_con_8<-cv_matrix_knn(train,mandmatrix,8)
man_con_14<-cv_matrix_knn(train,mandmatrix,14)


#Q9
# combine training and testing data
total_image<-rbind(train,test)
# calculate Euclidean distance matrix for all 6000 images
eucdmatrix6<-dist(total_image[,-1],upper=T,diag=T)
eucdmatrix6<-as.matrix(eucdmatrix6)
# save the distance matrix as a RDS file
saveRDS(eucdmatrix6,"D:/R/FP/eucdmatrix6.rds")
# calculate Manhattan distance matrix for all 6000 images
mandmatrix6<-dist(total_image[,-1],upper=T,diag=T,method="manhattan")
mandmatrix6<-as.matrix(mandmatrix6)
# save the distance matrix as a RDS file
saveRDS(mandmatrix6,"D:/R/FP/mandmatrix6.rds")

# read saved distance matrices
eucdmatrix6<-readRDS('eucdmatrix6.rds')
mandmatrix6<-readRDS('mandmatrix6.rds')

# initialize error rate vectors
test_error_rate1<-numeric()
test_error_rate2<-numeric()
#use train data to predict test data
# for each k value
for(k in 1:15){
  # calculate error rate for test data set using k value and Euclidean and Manhattan distance
metrics
  test_error_rate1[k]<-
sum(knn_predict(test,train,eucdmatrix6[5001:6000,1:5000],k)!=test[,1])/nrow(test)
  test_error_rate2[k]<-
sum(knn_predict(test,train,mandmatrix6[5001:6000,1:5000],k)!=test[,1])/nrow(test)
}
#data cleaning
# generate sequence of numbers 1 through 15 for each k-value
k<-seq(1,15,1)
# bind the error rates for euclidean and manhattan distance matrices together with the k value
test_error_rate<-data.frame(cbind(test_error_rate1,test_error_rate2,k))
# give column names to data frame
```

```
colnames(error_rate)<-c("Euclidean","Manhattan","k")
test_error_rate<-melt(test_error_rate,id="k")
levels(test_error_rate$variable)<-c('Euclidean','Manhattan')
#display the test error rate
ggplot(data=test_error_rate,aes(x=k,y=value,col=variable,label=value))+
  geom_point()+geom_line()+
  scale_x_continuous(breaks = c(seq(1,15,1)))+
  geom_label()+
  ylab("error_rate")+ggtitle("test_error_rate for different k and metric combination")+
  theme_classic()
```