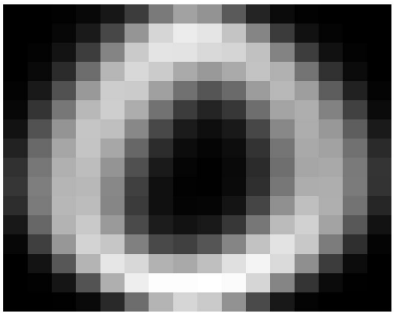
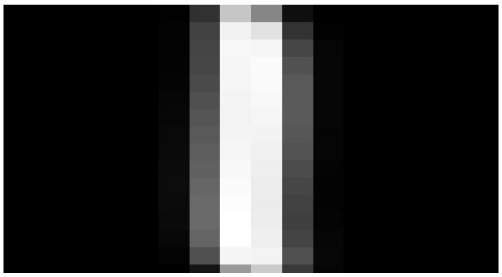
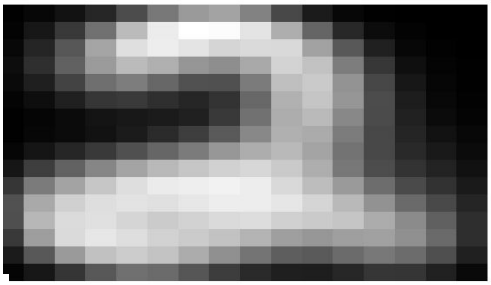
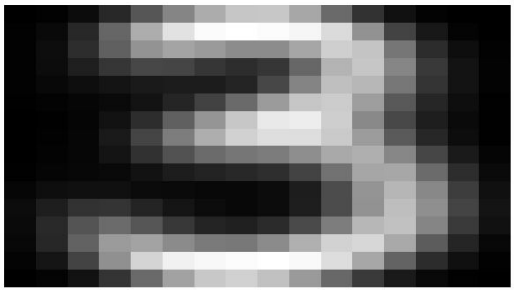
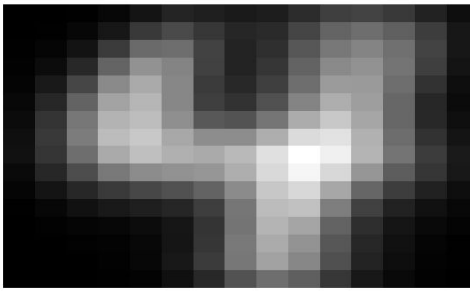
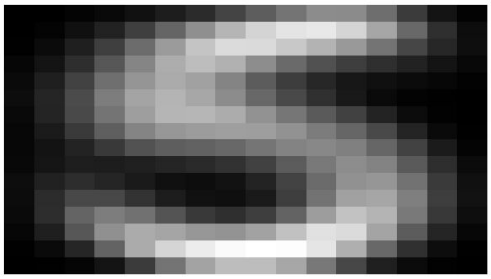
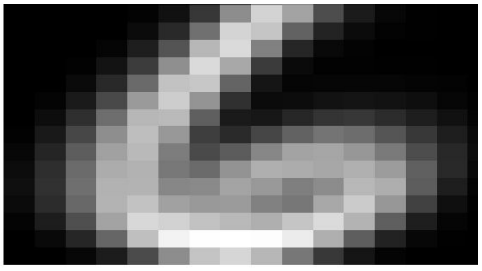
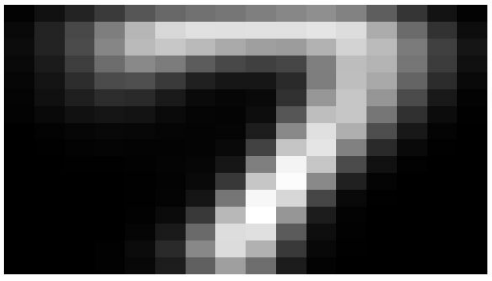
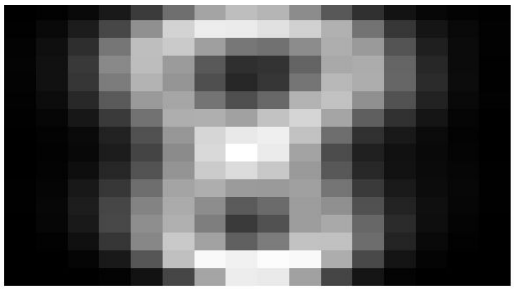
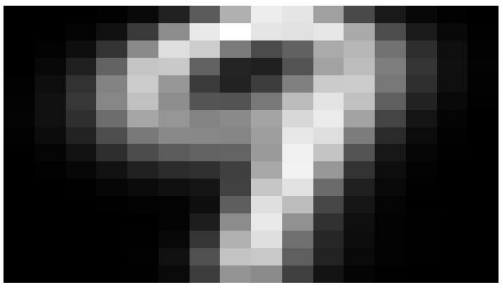
Group 52:
Mitchell Layton,
Caroline Mai,
William Powell

STA 141A - Final Project

In this project, we use data concerning pixels to classify handwritten digits in zip codes for outgoing mail for the United States postal service. The data consists of files which have information on the observations, which contain class labels for each digit 0 to 9, and 256 pixel values that constitute a 16x16 gray scale image of the digit.

3) (a) Below are the images for what each digit, 0 through 9, looks like on average.

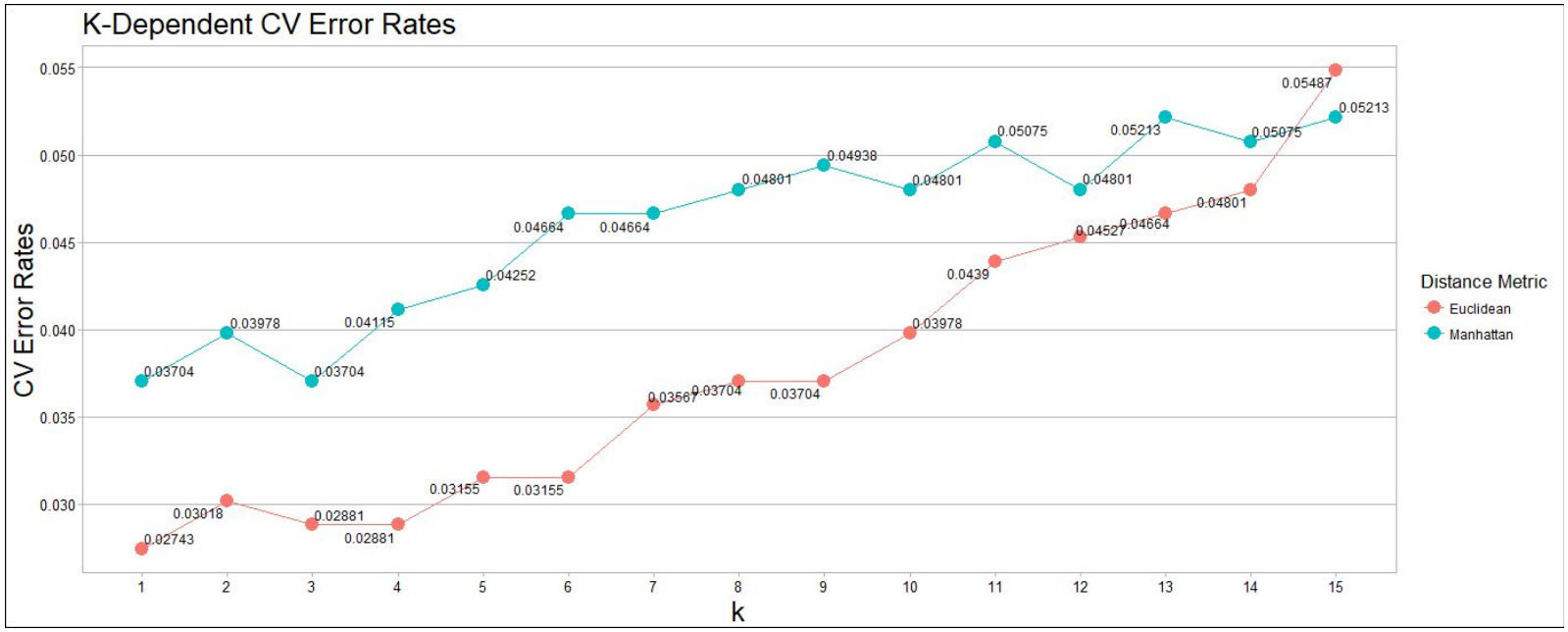












(b) We next would look to determine how important certain pixels are for classification. To do so, we look at the variance for each pixel between observations. The pixels with the highest variance between observations would be most likely to be useful for classification, while the pixels with the lowest variance between observations would not be as useful for classification. We find that the most useful are pixels 231, 220, and 106 with variances 0.7995, 0.7786, and 0.7761 respectively. The least useful are pixels 242, 2, and 257 with respective variances 0.00222, 0.00267, and 0.00436.

5) Despite putting our predict_knn() function inside a for loop, I think our methodology of creating the folds and indexing the observations and partitioning the folds was successful and efficient. Obviously, it is not super efficient to have our cv_error_knn() function call predict_knn() for each of the 10 partitioned folds because of the distance calculations, but we found this method to be our only way of proceeding with the function. Furthermore, instead of base R dist() function, we took some advice offered on Piazza and used the "parallelDist" library in which we used parDist() as a more efficient means of calculation distances. This improved our efficiency because the function computes distances of the matrices in parallel using multiple threads.

6)



We tested 10-fold cross validation error rates for k's one through fifteen for both the Euclidean and Manhattan methods of calculating distance. There appears to be a positive relationship between k and error rates meaning as k increases, so does the error rate. From our results, we consistently see that the Euclidean method results in a lower error rate than the Manhattan method for all k except for k = 15. By

the way the distances are calculated, the Euclidean method should always be less than doing so with the Manhattan method. Nonetheless, k = 1, 3, 4 resulted in the lowest error rates for both methods. Since error rates look like they increase as k increases, it might not be useful to consider additional values of k.

7) The following are confusion matrices for the k & metric combinations that yielded the lowest rates. A confusion matrix shows the frequencies of predicted class labels versus the true class labels found in the data set. The values found in the diagonal are those that were accurately predicted, while those not found in the diagonal were inaccurate predictions.

K = 1, Euclidean method

	prediction									
real	0	1	2	3	4	5	6	7	8	9
0	1187	0	4	0	0	1	2	0	0	0
1	0	1002	0	0	2	0	0	1	0	0
2	4	3	697	6	0	1	2	14	3	1
3	4	0	6	633	0	9	0	0	5	1
4	3	5	3	1	620	0	2	2	1	15
5	7	0	4	12	2	520	5	1	3	2
6	8	1	0	0	0	3	651	0	1	0
7	0	2	1	0	4	1	0	628	1	8
8	3	5	2	14	1	6	4	3	503	1
9	1	0	0	1	8	0	0	9	1	624

K = 3, Euclidean method

	prediction									
real	0	1	2	3	4	5	6	7	8	9
0	1183	0	4	2	0	1	4	0	0	0
1	0	1003	1	0	0	0	0	1	0	0
2	6	2	694	8	0	1	2	11	4	3
3	2	1	1	636	1	8	0	0	8	1
4	1	11	3	0	608	1	6	3	0	19
5	7	1	4	7	3	527	4	0	1	2
6	13	1	0	0	1	2	645	0	2	0
7	0	3	1	0	4	0	0	629	0	8
8	5	5	2	9	2	6	5	6	500	2
9	1	0	0	1	8	0	0	11	1	622

K = 4, Euclidean method

prediction										
real	0	1	2	3	4	5	6	7	8	9
0	1183	0	4	2	0	1	4	0	0	0
1	0	1003	1	0	0	0	0	1	0	0
2	8	3	692	5	3	1	1	12	5	1
3	3	1	1	637	1	6	0	1	7	1
4	1	14	4	0	605	0	4	2	0	22
5	10	0	4	7	3	523	6	0	0	3
6	10	1	0	0	0	4	647	1	1	0
7	0	3	1	0	6	1	0	623	0	11
8	4	6	1	14	4	6	3	4	496	4
9	2	1	0	1	7	0	0	13	0	620

Considering that each of these three confusion matrices have relatively similar error rates, we would want to make sure that we do not select one with significant bias against predicting any single digit. The most frequent incorrect classification among all three levels of k was to classify a digit 4 as 9. This is expected given the numbers' visual similarity, and this mistake occurred least often for k = 1, which is already our default choice. The most incorrectly predicted digit over all was 8, with error rates of 7.196%, 7.749%, and 8.487% for k = 1, 3, and 4 respectively. Again, this does not change which combination we would choose as best, since k = 1 has the lowest error rate for digit 8, and the lowest error rate for all digits.

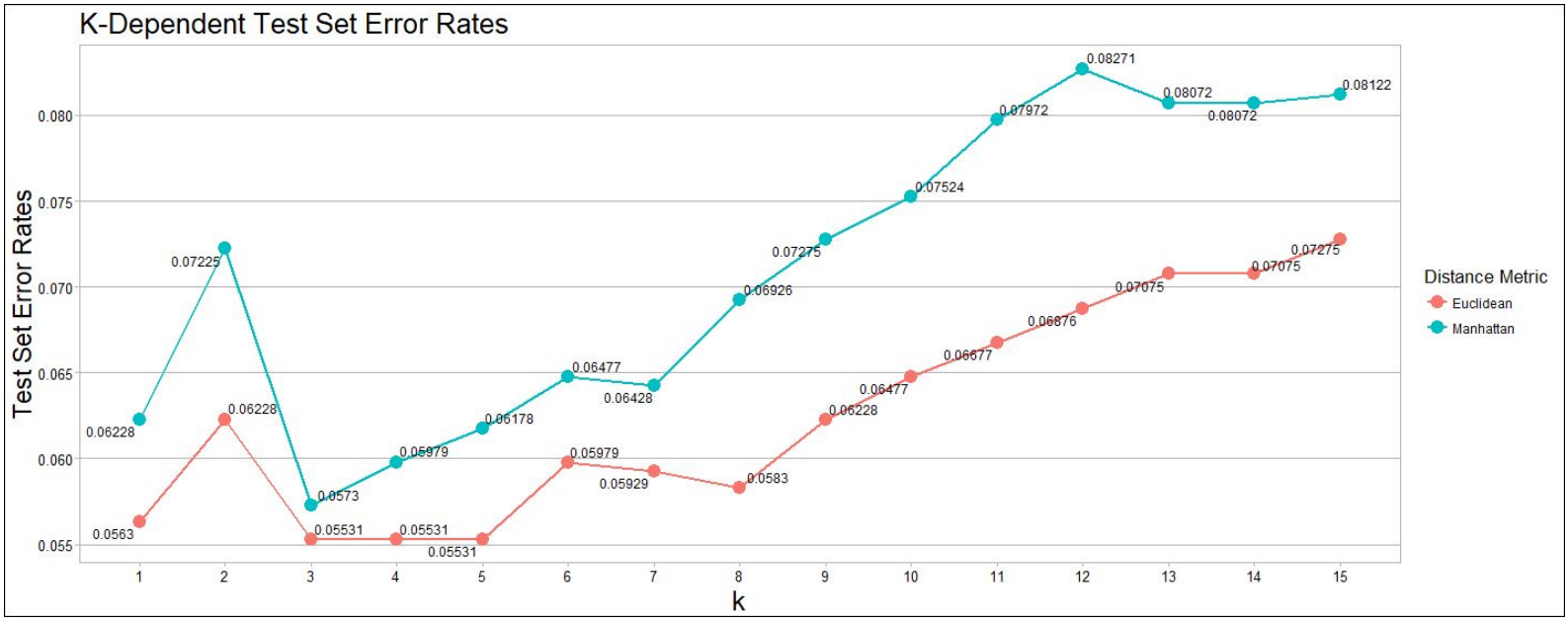
8)

The confusion matrix of our best k and distance metric combination (k = 1, Euclidean) implies several things about the k-NN classifier. In general, the digits that occurred more frequently in the training data were more often classified correctly, because they had a higher contribution to calculating the predictions. The most "confused" digits were:

● 4 and 9 (23 incorrect)	● 3 and 5 (21 incorrect)
● 3 and 8 (19 incorrect)	● 7 and 9 (17 incorrect)

Judging by these, it seems that the classifier was bad at dealing with geometrically similar digits. This is to be expected, since our criteria for k-nn was distance, but a more complex classifier could potentially overcome this weakness.

9)



The above plot presents test set error rates for k = 1, … 15 using the Euclidean and Manhattan methods . It follows a similar trend as the 10-fold cross validation rates for k = 1,...15 using the same methods- increasing error rate as k increases. However, the test set error rates were all higher than that for cross validation, starting at around 0.05 versus 0.02 for k = 1. Interestingly enough, there was a spike at k = 2 across methods of calculating distance and type of error before decreasing at k = 3 and continuing the constant increasing trend. The test set error rate may be closer to the "true" error rate since it uses the test data the model wasn't trained on. Still, the test set error rate is relatively low for the k's we selected which shows that our knn-classifier achieved a good level of accuracy.

10) As a group, we collaborated and delegated responsibilities through Slack and we experienced some overlapping between contributions to problems.
Mitchell: Contributed to #1, #2, #3(a), #5, #6, wrote cnn_error_knn based off William's #4 predict function, displayed digits and cv error graph.
William: Contributed to #4, #7, and #8, and helped with finding and plotting some of the error rates.
Caroline: Contributed to #3b, #6, #7, and #9 including interpretations in report.

Code Appendix

```
# STA 141A - Final Project
# Due December 5, 2017 @ 5:00 PM
# GROUP 52: Mitchell Layton, Caroline Mai, William Powell
#-------------------------------------------------------
library(ggthemes)
library(reshape)
library(ggplot2)
library(parallelDist)
#1)---------------------------------------------------------------------
read_digits = function(x) {
   txt = read.table(x)
}

train = read_digits("PATH_OF_train.txt")
test = read_digits("PATH_OF_test.txt")

#2)---------------------------------------------------------------------

view_digit = function(read_data, observation) {
   temp = melt(read_data[observation,])
   temp = temp$value[2:length(temp$value)]
   temp = matrix(temp, nrow = 16, ncol = 16, byrow = T)
   rotate <- function(x) {
      t(apply(x, 2, rev))
   }
   temp = rotate(temp)
   image(temp, axes = FALSE, col = grey(seq(0, 1, length = 256)))

}

# Put desired dataset and observation in parameters
view_digit(train,130)

#3)---------------------------------------------------------------------

# Display  graphically what each digit (0-9) looks like on average

avg_digit = function(read_data) {

   rotate <- function(x) {
      t(apply(x, 2, rev))
   }

   nums = seq(0,9,1)
   for (i in nums) {
```

```
    # Used sqldf() package for SQL query like statements. Also to get sqldf to use my i in my for loop,
    # had to use sprintf and strwrap for wrapping my SQL statement string sentence to be able to
    # use the C-style string formatting command within the sqldf() function.
    temp = sqldf(strwrap(sprintf("SELECT * FROM read_data WHERE V1 = '%s'",i), simplify = T))
    temp = as.data.table(sapply(temp,mean))[2:257]
    temp = melt(temp, id=1)
    temp = temp$V1
    temp = matrix(temp, nrow = 16, ncol = 16, byrow = T)
    temp = rotate(temp)
    print(image(temp, axes = FALSE, col = grey(seq(0, 1, length = 256))))
  }

}
avg_digit(data)

#4)--------------------------------------------------------------------
start_time <- Sys.time()
predict_knn = function(points, train, k, metric) {

  #Set up a vector for the predicted classes
  predictions = c(rep(NA,nrow(points)))

  #A matrix of the prediction points, excluding the class values, to use for the distance function
  dist_mat = as.matrix(rbind(points,train[1:nrow(train),])[,2:257])

  #A matrix of all the distances
  one_dist = as.matrix(parDist(dist_mat,method = metric))

  for (i in 1:nrow(points)){
    #frequencies of the nearest classes stored in "classes"
    classes = c(rep(0,10))
    #Extracting distances from the matrix
    distances = one_dist[(1+nrow(points)):nrow(one_dist),i]
    #Finding the closest k
    k_nearest = sort(distances)[1:k]
    for(j in 1:k){
      #incrementing frequencies of classes for the k-nearest neighbors
      classes[train[which(distances==k_nearest[j]),1] + 1] = classes[train[which(distances==k_nearest[j]),1] + 1] +
1
    }
    #print(classes)

    #Solving ties
    if (length(which(classes==max(classes))) > 1){
      #choose one of the max frequency classes at random (-1 to adjust from 1:10 to 0:9)
      predictions[i] = sample(which(classes==max(classes)),1) - 1
    }
    else{
```

```
        #choose the max frequency class
        predictions[i] = which(classes==max(classes)) - 1
      }
    }
    #output the vector of predictions
    predictions
}
#Or try it on the test data
#predict_knn(test,train,95,"euclidean")
end_time <- Sys.time()
end_time - start_time

#5)----------------------------------------------------------------------

cv_error_knn = function(train_data,k,metric) {

    #Create 10 equally size folds
    folds <- cut(seq(1,nrow(train_data[2:257])),breaks = 10,labels = FALSE)

    #Perform 10 fold cross validation
    for(i in 1:10) {
        #Segment your data by fold using the which() function
        test_indexes <- which(folds == i,arr.ind = TRUE)
        tested = train_data[test_indexes, ]
        compare = tested$V1
        trained = train_data[-test_indexes, ]

        #Use the test and train data partitions:
        P = as.data.table(predict_knn(tested,trained,k,metric))
        x = length(which(P!=compare))
        y = x/length(compare)
        y
    }
    m = mean(y)
    return(m)
}
# Put appropriate k-values and distance metric in function parameters below
cv_error = cv_error_knn(train,1,"euclidean")
cv_error


#6)----------------------------------------------------------------------

rates1 = c(rep(0,15))
rates2 = c(rep(0,15))

start_time <- Sys.time()
for (i in 1:15) {
```

```
  error_rates_EUC = cv_error_knn(train,i,"euclidean")
  error_rates_MAN = cv_error_knn(train,i,"manhattan")

  rates1[i] = error_rates_EUC
  rates2[i] = error_rates_MAN

}
end_time <- Sys.time()
end_time - start_time

x = seq(1,15,1)
data = as.data.frame(cbind(x,rates1,rates2))
names(data) = c("k","Euclidean","Manhattan")
data = melt(data, id = "k")
names(data) = c("k","Distance Metric","CV_Error")

p = ggplot(data) +
  geom_point(aes(x = k, y = CV_Error, colour = `Distance Metric`), size = 4, shape = 19) +
  geom_line(aes(x=k,y=CV_Error, colour = `Distance Metric`)) +
  geom_text_repel(aes(x=k,y=CV_Error,label=round(CV_Error,5)),size=3.15) +
  theme_calc() +
  scale_x_continuous("k", breaks = c(seq(1,15,1))) +
  scale_y_continuous("CV Error Rates") +
  labs(title = "K-Dependent CV Error Rates") +
  theme(axis.title.x = element_text(size = rel(1.25))) +
  theme(axis.title.y = element_text(size = rel(1.15))) +
  theme(axis.text.x = element_text(size = rel(1.25))) +
  theme(axis.text.y = element_text(size = rel(1.25))) +
  theme(title = element_text(size = rel(1.5))) +
  theme(legend.title = element_text(size = rel(.85))) +
  theme(legend.text = element_text(size = rel(.9)))

p

#7)----------------------------------------------------------------------

create_errormatrix = function(train_data,k,metric) {

  #Create 10 equally size folds
  folds <- cut(seq(1,nrow(train_data[2:257])),breaks = 10,labels = FALSE)

  #Empty vectors to append the predictions and real values
  all_P = c()
  all_real = c()
  #Perform 10 fold cross validation
  for(i in 1:10) {
    #Segment your data by fold using the which() function
```

```
    test_indexes <- which(folds == i,arr.ind = TRUE)
    tested = train_data[test_indexes, ]
    all_real = c(all_real,tested$V1)
    trained = train_data[-test_indexes, ]

    #Use the test and train data partitions:
    P = predict_knn(tested,trained,k,metric)
    all_P = c(all_P,P)
  }
  #Use all the appended values in the matrix
  return(data.frame(real = all_real, prediction = all_P))
}
#The data frame now has all 7291 obs.


start_time <- Sys.time()
for (i in c(1,3,4)) { #3 best K-values and dist metric

  testing = create_errormatrix(train,i,"euclidean")
  print(table(testing))

}
end_time <- Sys.time()
end_time - start_time

#8)--------------------------------------------------------------------
# Analysis In Report.

#9)--------------------------------------------------------------------

start_time <- Sys.time()

actual_test = test[,1]
test_frame = as.data.frame(actual_test)
test_eu = c(rep(0,15))
test_man = c(rep(0,15))

for (i in 1:15) {

  pred_eu = predict_knn(test,train,i,"euclidean")
  pred_man = predict_knn(test,train,i,"manhattan")

  eu_frame = cbind(pred_eu, test_frame)
  man_frame = cbind(pred_man, test_frame)

  error_eu = nrow(subset(eu_frame, pred_eu!= actual_test))/nrow(test)
  error_man = nrow(subset(man_frame, pred_man!= actual_test))/nrow(test)
```

```
  test_eu[i] = error_eu
  test_man[i] = error_man

}
end_time <- Sys.time()
end_time - start_time


x = seq(1,15,1)
data2 = as.data.frame(cbind(x,test_eu,test_man))
names(data2) = c("k","Euclidean","Manhattan")
data2 = melt(data2, id = "k")
names(data2) = c("k","Distance Metric","test_set_Error")
p2 = ggplot(data2) +
  geom_point(aes(x = k, y = test_set_Error, colour = `Distance Metric`), size = 4, shape = 19) +
  geom_line(aes(x = k, y = test_set_Error, colour = `Distance Metric`), size = 1) +
  geom_text_repel(aes(x=k,y=test_set_Error,label=round(test_set_Error,5)),size=3.15) +
  theme_calc() +
  scale_x_continuous("k", breaks = c(seq(1,15,1))) +
  scale_y_continuous("Test Set Error Rates") +
  labs(title = "K-Dependent Test Set Error Rates") +
  theme(axis.title.x = element_text(size = rel(1.25))) +
  theme(axis.title.y = element_text(size = rel(1.15))) +
  theme(axis.text.x = element_text(size = rel(1.25))) +
  theme(axis.text.y = element_text(size = rel(1.25))) +
  theme(title = element_text(size = rel(1.5))) +
  theme(legend.title = element_text(size = rel(.85))) +
  theme(legend.text = element_text(size = rel(.9)))

p2
```