Introduction –

Problem Description: This report deals with the basic level image processing problem. It classifies handwritten zip codes by using K-nearest neighbors (KNN) algorithms. The main objective is to be able to find a model that accurately predicts the dependent variable (limited to digit 2 or 7 in this context) on a new dataset. In summary, the steps used to achieve the objective are: build classification models using training dataset, evaluate model performance on training and testing datasets, refine tuning parameters and evaluate robustness of models using cross-validation.

Data Set: The dataset used in this project contains digital information of handwritten zip codes scanned from envelopes of the U.S. Postal Service. The dataset is normalized where the original digits' orientation and size were adjusted to create recognizable and comparable 16x16 grayscale images. The first column of the dataset contains the "dependent variable" - sorted digital numbers from 0-9. The other 256 variables in the dataset contains grayscale values of the images, ranging from 0 to 255 where each number defines a color. In this case, 0 means black and 255 means white and every other number in between is a shade of gray ranging from black to white. Source of dataset: https://hastie.su.domains/ElemStatLearn/datasets/zip.info.txt

Exploratory Data Analysis:

After filtering on digits 2's and 7's on the original training and testing datasets, the distribution turned out to be similar between 2's and 7's with 2's being slightly higher. The distribution also looks equivalent while compared between the training and testing datasets. The two tables show distribution by number of observation and by proportions respectively.

	Total obs	2's	7's
train	1376	731	645
test	345	198	147

	Total obs	2's	7's
train	1376	53%	47%
test	345	57%	43%

With a very high number of independent variables, driving meaning out of the results of summary statistics is difficult but they seem to have a similar kind of patterns. Same is the case for correlation matrix results. The results of these analysis can be viewed by running the provided code in the appendix.

The dataframe is converted into 16x16 matrix to visualize the main image data. For example, the images of the 5th and 15th rows turns out to be 7 and 2 respectively. Grayscales values of 0 and 1 show crisp black and white images of 7 and 2.

Methods/Methodology –

The two classification methods used to build the classification rules are: KNN regression and KNN classification.

KNN regression:

A linear regression model was built using the training dataset and then used to predict the Y variable on the training dataset which was written in the form to classify the prediction value as 7 if the predicted value is greater than or equal to 4.5 (which is the 50% of the data i.e., 2+7/2=4.5) and 2 otherwise.

KNN classification:

In the KNN classification context, Y is classified based on the majority vote of the K nearest neighbors of the training data using the function knn from R package class. Parameter k was tested using eight different values: [1, 3, 5, 7, 9, 11, 13, 15]. This was to check which number of k produced the smallest error or the largest accuracy to optimize the performance of the model.

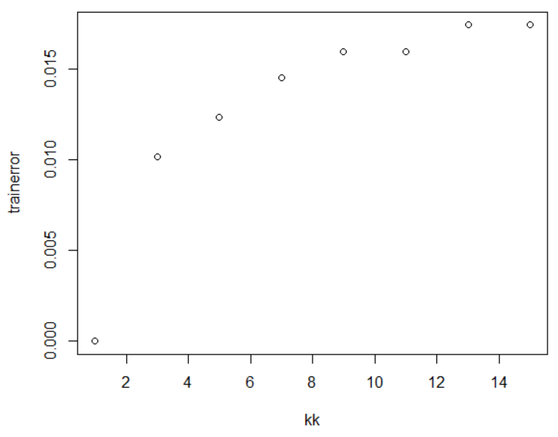
Training errors and Testing errors:

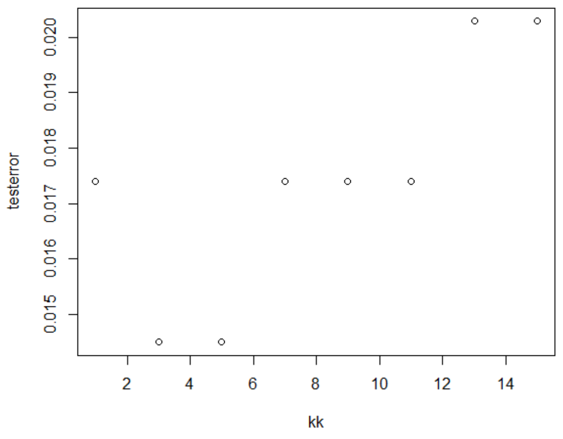
Once the regression and knn models were built, they were applied on both training and testing datasets to evaluate the predictive power of each model by calculating error and accuracy. The error is calculated as the average of the predicted Y variables that are not closest to the actual Y values and accuracy is simply the opposite (i.e., predicted = actual Y values). The following two tables show training and testing errors and training and testing accuracies respectively as a percent. There are visual representations of the same data for the knn classifications model in the charts below shown as rates.

	training error	**testing error**
Linear Regression	0.1%	1.7%
KNN1	0.0%	1.7%
KNN3	1.0%	1.4%
KNN5	1.2%	1.4%
KNN7	1.5%	1.7%
KNN9	1.6%	1.7%
KNN11	1.6%	1.7%
KNN13	1.7%	2.0%
KNN15	1.7%	2.0%

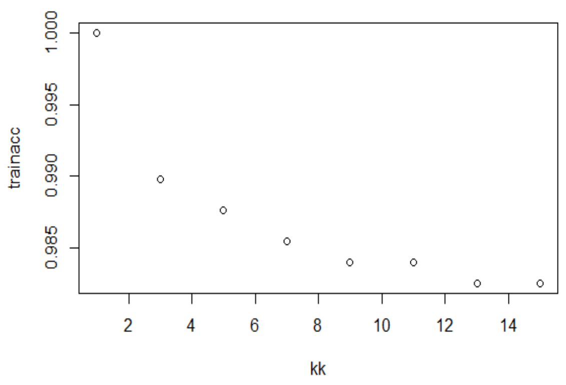
	training accuracy	**testing accuracy**
Linear Regression	99.9%	98.3%
KNN1	100.0%	98.3%
KNN3	99.0%	98.6%
KNN5	98.8%	98.6%
KNN7	98.5%	98.3%
KNN9	98.4%	98.3%
KNN11	98.4%	98.3%
KNN13	98.3%	98.0%
KNN15	98.3%	98.0%

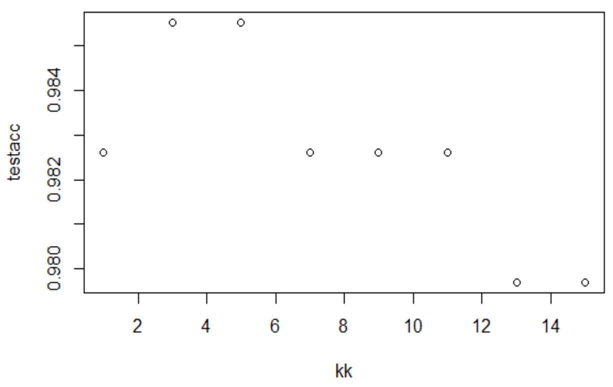
Knn errors:





Knn accuracy:





Based on the results above, knn=1 seem to have the lowest error rate or the highest accuracy rate of 100%. Based on the train error vs kk graph, the error rate seems to be increasing as the number of k increases but not significantly. However, the testing error for knn=1 of 1.7% is not the lowest among all different values of k. The testing error is known to be larger than the training error since the model was built using the training dataset, so could be causing some overfitting.

Comparing knn=1 model and linear regression model, the average training error for linear regression (0.1%) is only slightly worse than knn=1's perfect average error rate of 0.0%. However, the testing error of linear regression is also higher than its training error but matches with knn=1's testing error (1.7%).

Cross-Validation

Both training and testing datasets were small, thus they were combined for the purpose of cross-validation. Monte Carlo Cross Validation was performed to further assess the performance of each model. In the CV process, the combined dataset was randomly split into training (80%) and testing (20%) subsets in each of the 100 runs for linear regression and 8 knn models. In each loop, training subset was used to build the model and the testing subset was used to evaluate their performances. This resulted into a total of 100 testing errors calculated for each of the 9 models (1 regression and 8 knn). For each model, mean and variances of the 100 testing errors were calculated as below.

Findings/Conclusions –

	Error Mean	Error Variance	Accuracy Mean
Linear Regression	1.12%	3.08322E-05	98.9%
KNN1	1.26%	2.27531E-05	98.7%
KNN3	1.37%	2.17389E-05	98.6%
KNN5	1.53%	2.73018E-05	98.5%
KNN7	1.64%	2.9061E-05	98.4%
KNN9	1.74%	0.000030805	98.3%
KNN11	1.86%	3.46553E-05	98.1%
KNN13	1.95%	3.56567E-05	98.1%
KNN15	2.01%	4.42688E-05	98.0%

After running cross-validation for 100 times for each of the 9 models, the linear regression seems to be the best performing model. Almost all of 9 models are strong to predict the zip code digits (2's and 7's) very well given the >98% accuracy rate, very low rates of error and almost non-existent variances. Among the knn models, all 8 different values of k seem to be optimal tuning parameters but by a slight margin, k=1 seems to have the lowest mean error of 1.26% and the highest accuracy of 98.7%. The pattern of error increasing with the increased number of values of k still hold true. Comparing the error and accuracy rates of 1) linear regression w/o CV, 2) KNN [1, 3, 5, 7, 9, 11, 13, 15] w/o CV, 3) linear regression with 100 CV, and 4) KNN [1, 3, 5, 7, 9, 11, 13, 15] with 100 CV, I would use the knn=1 with 100 CV model in the real-world scenario because of the discrete nature of the dependent variable. If we were predicting a continuous variable, then a linear regression model could have been a stronger choice. For future reference, these models can be improved in different ways to improve the robustness and efficiency further. We can use a bigger dataset that could prevent the issue of overfitting caused by higher variability. The number of independent variables can also be reduced using various approaches to prevent any biases. The cutoff used to categorize the predicted value of linear regression as 7 and 2 can also be better optimized. Lastly, we could also expand the dependent variable to include other digits of the zip codes.

Appendix

R code

```
setwd('C:/Users/stuti/Desktop/ISYE7406_Spring2023/Module 1/HW 1')


############ 1. Read Training data ############

ziptrain <- read.table(file="zip.train.csv", sep = ",");

ziptrain27  <-  subset(ziptrain,  ziptrain[,1]==2  |  ziptrain[,1]==7);
#limit to 2's and 7's


## some sample Exploratory Data Analysis

dim(ziptrain27); ## 1376 257

sum(ziptrain27[,1] == 2); #2=731, 7=645

summary(ziptrain27); #summary stats

round(cor(ziptrain27),2); #coorelation



## To see the letter picture of the 5-th row by changing the row
observation to a matrix

rowindex = 5; ## You can try other "rowindex" values to see other rows

ziptrain27[rowindex,1];

Xval                =                t(matrix(data.matrix(ziptrain27[,-
1])[rowindex,],byrow=TRUE,16,16)[16:1,]);

image(Xval,col=gray(0:1),axes=FALSE) ## Also try "col=gray(0:32/32)"

image(Xval,col=gray(0:32/32),axes=FALSE)

#


####################      2.      Build      Classification      Rules
#######################

### linear Regression

mod1 <- lm( V1 ~ . , data= ziptrain27);

pred1.train <- predict.lm(mod1, ziptrain27[,-1]);
```

```
y1pred.train <- 2 + 5*(pred1.train >= 4.5); ## depending on the indicator
variable whether pred1.train >= 4.5 = (2+7)/2.



## Note that we predict Y1 to 2 and 7
mean( y1pred.train != ziptrain27[,1]); #training error
mean( y1pred.train == ziptrain27[,1]); #training accuracy

### knn Classification

library(class);
kk <- seq(1,15,2); # for 8 different k values
trainerror = rep(x = 0, times = length(kk))
trainacc = rep(x = 0, times = length(kk))
for (i in 1:length(kk)){
  xnew <- ziptrain27[,-1];
  ypred.train <- knn(ziptrain27[,-1], xnew, ziptrain27[,1], k = kk[i]);
  trainerror[i] <- mean(ypred.train  != ziptrain27[,1]); #training error
  trainacc[i]  <-  mean(ypred.train    ==  ziptrain27[,1]);  #training
accuracy

}
plot(kk, trainerror)
#print(trainerror)
plot(kk, trainacc)
#print(trainacc)

######################## Testing Error ######################
### read testing data
ziptest <- read.table(file="zip.test.csv", sep = ",");
ziptest27 <- subset(ziptest, ziptest[,1]==2 | ziptest[,1]==7);
```

```
dim(ziptest27) ##345 257

sum(ziptest27[,1] == 7); #2=731, 7=645


### testing error for regression classification rule

pred1.test <- predict.lm(mod1, ziptest27[,-1]);

y1pred.test <- 2 + 5*(pred1.test >= 4.5); ## depending on the indicator
variable whether pred1.train >= 4.5 = (2+7)/2.


## Note that we predict Y1 to 2 and 7

mean( y1pred.test != ziptest27[,1]); #testing error

mean( y1pred.test == ziptest27[,1]); #testing accuracy


## Testing error of KNN, and you can change the k values.

library(class);

kk <- seq(1,15,2);

testerror = rep(x = 0, times = length(kk));

testacc = rep(x = 0, times = length(kk));

#cverror <- rep(x = 0, times = length(kk));

for (i in 1:length(kk)){

  xnew2 <- ziptest27[,-1];

  ypred.test <- knn(ziptrain27[,-1], xnew2, ziptrain27[,1], k = kk[i]);

  testerror[i] <- mean(ypred.test  != ziptest27[,1]); #testing error

  testacc[i] <- mean(ypred.test  == ziptest27[,1]);#testing accuracy


}


plot(kk, testerror)

#print(testerror)

plot(kk,testacc)

#print(testacc)
```

```
########################              4.              Cross-Validation
#####################################

zip27full = rbind(ziptrain27, ziptest27) ### combine to a full data set

n1 = 1376; # training set sample size

n2= 345; # testing set sample size

n = dim(zip27full)[1]; ## the total sample size

set.seed(7406); ### set the seed for randomization




### Initialize the TE values for all models in all $B=100$ loops

B= 100; ### number of loops

TEALL = NULL; ### Final TE values

for (b in 1:B){

  ### randomly select n1 observations as a new training subset in each
loop

  flag <- sort(sample(1:n, n1));

  #print(flag)

  zip27traintemp <- zip27full[flag,]; ## temp training set for CV

  zip27testtemp <- zip27full[-flag,]; ## temp testing set for CV


  library(class);

  kk =7


  cverror <- NULL;


  for (i in 0:kk){

    xnew3 <- zip27testtemp[,-1];

    kk <-2*i+1
    #regression

    mod.fulldata <- lm( V1 ~ . , data= zip27traintemp);

    pred.test.fulldata <- predict.lm(mod.fulldata, zip27testtemp[,-1]);

    y1pred.fulldata <- 2 + 5*(pred.test.fulldata  >= 4.5);

    te0  <-  mean(  y1pred.fulldata   !=  zip27testtemp[,1]);  #regression
testing error

    #print(te0)



    #knn

    ypred.fulldata       <-       knn(zip27traintemp[,-1],       xnew3,
zip27traintemp[,1], k = kk);

    testerror <- mean(ypred.fulldata != zip27testtemp[,1]); #knn testing
error

    cverror <- cbind(cverror, testerror);

    #print(cverror)

  }

  TEALL = rbind( TEALL, cbind(te0,cverror));

  #print(TEALL)

}


dim(TEALL); ### 100X9 matrices


colnames(TEALL) <- c( "linearRegression","KNN1", "KNN3", "KNN5", "KNN7",

                      "KNN9", "KNN11", "KNN13", "KNN15");

## You can report the sample mean/variances of the testing errors so as
to compare these models

apply(TEALL, 2, mean);

apply(TEALL, 2, var);


##end##
```