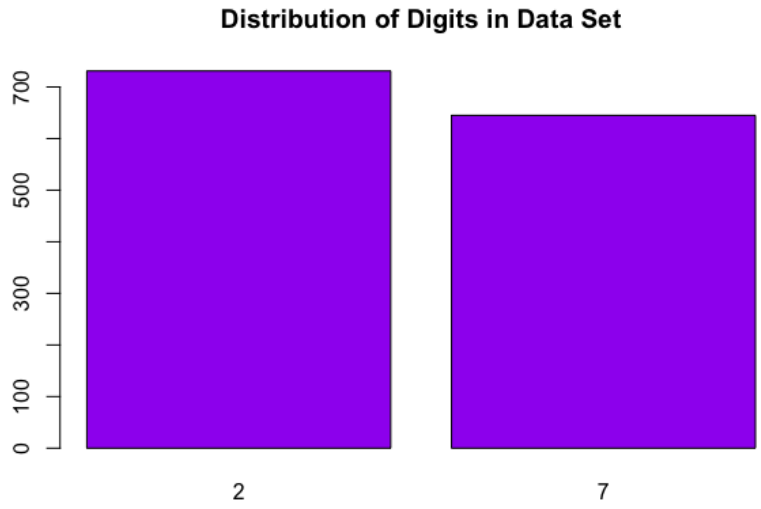
Introduction

The overarching goal of this homework assignment is to do analysis on and classify data from the zip code data set. This data set consists of grayscale images of handwritten digits. The first column represents the response variable, in this case the digit, which ranges from 0 to 9. The rest of the columns are the pixel intensity values which make up the digit images. Each image comes from a 16 x 16 grid.

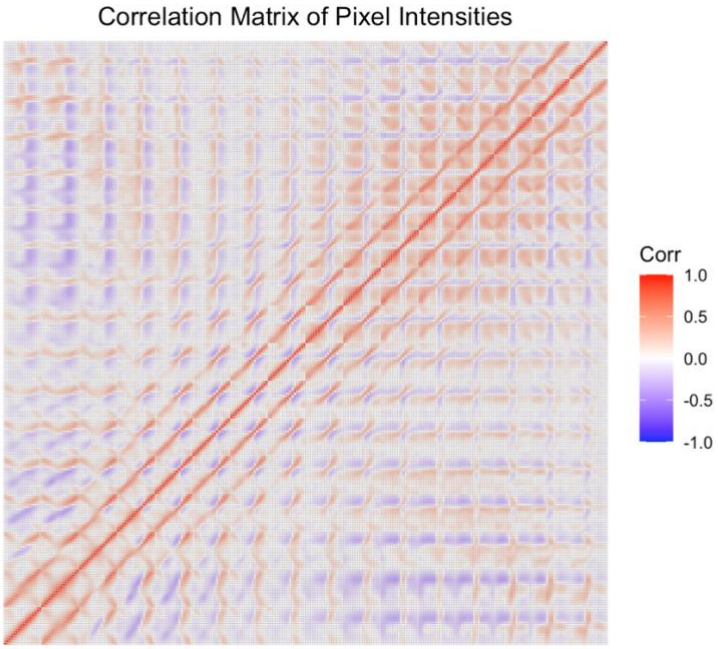
The specific problem of this homework is a binary classification task involving distinguishing between two handwritten digits, 2 and 7, based on pixel intensity values. So, a subset of the zip code data set will be taken where the response variable is equal to 2 or 7 and analysis will be done on it.

Exploratory Data Analysis

The size of the data set including only the response variables 2 and 7 is 257 columns long. The first column is the response variable, and all the others are the 256 independent pixel intensities for each response variable. The total number of observations of the data set is 1,376. The total number of response variables that are the number 2 is 731 and the total number of 7's is 645. So, there are 86 more 2's than 7's.

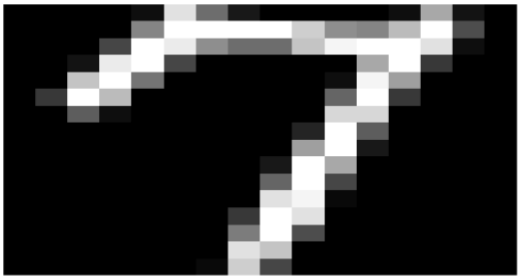


A correlation matrix was built to show how the pixel values correlate with each other. The response variable was left out because correlation is generally for continuous variables and not categorical ones. The following graph shows the matrix:



This graph shows pairwise correlations between variables, for example, (the second column with the second column), (the second column with the third column), (the second column with the fourth column), etc. It then color codes that specific pair with a shade of red if positively correlated, white if they are not correlated, and a shade of blue if they are negatively correlated. For this data set, if pixels are positively correlated then they are likely in the same region of the image and have similar intensities (like one light and the other light), if they are 0 correlated then they have no relationship, and if they are negatively correlated then they are likely in different parts of the image and have different intensities (like one light and the other dark). The diagonal line across the middle of the graph is from each column correlating with itself which would result in a +1.0, or perfect, correlation. It seems that there is a good mix of positively and negatively correlated pairs.

The following is what each row of pixels output when put into an image:



So, this specific row's pixel intensity values outputted the response variable 7, whereas a row's pixel intensities could also output a 2 as shown:

Methods/Methodology

Two methods were used for classification. One was linear regression and the other was the K Nearest Neighbors (KNN) algorithm.

Linear regression for classification maps class labels using decision boundaries. The model was fit using the pixel values as predictors and the class labels, 2 or 7, as the response. The boundary used was the midpoint of 2 and 7, 4.5. Predicted classes from the linear regression model were turned into class labels using the 4.5 boundary with the following equation: $Predicted\ Class = a + (b - a) \cdot (predicted\ value\ meets\ threshold)$ or in this case, $Predicted\ Class = \ 2 + 5 \cdot (prediciton \geq 4.5)$. So, if the left side of the equation was ≥ 4.5, then the data point was classified as a 7, and if it was less than it then it was classified as a 2. Error measurements were found using the training error, testing error, and cross validation.

The KNN algorithm was used with k values of (1, 3, 5, 7, 9, 11, 13, 15). This algorithm works by finding the k nearest points in the data and predicts the class of the data point based on the majority class around it. For example, if k = 3 and the data point trying to be classified has two 7's and one 2 as its three nearest data points, then that data point will be classified as a 7. Error measurements were also found using the training error, testing error, and cross validation.

The cross validation used was Monte Carlo cross validation. This kind of cross validation randomly splits the data set, the data set being the combination of both the training and testing data sets, into training and testing data and trains the model on that random split. The testing error is then found and saved. That process is then repeated x number of times,

in this case it was repeated 100 times, and the errors are averaged to provide an estimate for the model's performance.

Results

The error measurements differed with each different iteration of testing them. Below were the training errors of the linear regression model and the KNN model with each value of k used:

Linear Regression Training Error:

0.0007267442

KNN Training Errors:

k value	Training Error
1	0.00000000
3	0.01017442
5	0.01235465
7	0.01453488
9	0.01598837
11	0.01598837
13	0.01744186
15	0.01744186

Looking at both models, the linear regression model had a lower training error than every k value except for one of them, which was when k = 1. At k = 1 the training error was 0, meaning that there was not one incorrectly classified response. Because this is just on the training error, it was likely that this perfect accuracy was a result of overfitting. As more neighbors are used for classification, the model's performance decreased likely due to more generalized decisions. After these training errors were looked at, the testing errors were then observed:

Linear Regression Testing Error:

0.0173913043478261

KNN Testing Errors:

k value	Testing Error
1	0.01739130
3	0.01449275
5	0.01449275
7	0.01739130

9	0.01739130
11	0.01739130
13	0.02028986
15	0.02028986

Using the testing data, each error value increased. The linear regression testing error was comparable to the KNN training error's higher k values, and is about the same value as the KNN testing error when k = 1. The KNN model's lowest error was also not at k = 1 anymore. Both k = 3 and k = 5 produced a lower testing error than k = 1 and every other k value used. Once again, the higher k values produced the largest error values. The final errors values looked at were the average testing errors using cross validation and their variances:

Linear Regression Cross Validation Mean Testing Error:

0.01144928

Linear Regression Cross Validation Testing Error Variance:

2.365601e-05

KNN Cross Validation Mean Testing Errors and Testing Error Variances:

k value	Mean Testing Errors	Testing Error Variances
1	0.01342029	3.032297e-05
3	0.01402899	2.846698e-05
5	0.01576812	3.858285e-05
7	0.01689855	4.023516e-05
9	0.01744928	4.310783e-05
11	0.01860870	4.840338e-05
13	0.01991304	5.408506e-05
15	0.02072464	5.946887e-05

The average testing error using cross validation with linear regression was lower than the testing error without using it. The variance was also low, which indicates that the error was consistent across all 100 iterations. Similar to the pattern in the training errors with the KNN model, as the k value increased using the cross validation so did the mean testing error. The mean testing error at k = 1 was the lowest and was the highest at k = 15. Variances also increased as the k value increased, except for when the k value went from k = 1 to k = 3 the variance decreased. A higher variance at the larger k values indicates that there might be less consistent performance with the model.

Based on the cross validation mean testing errors and variances, the "optimal" choice of the tuning parameter k in the KNN model would be k = 3. With the second lowest average testing error and the lowest testing error variance, k = 3 provides a tradeoff in overfitting, which k = 1 would be the most overfit, and underfitting, the higher the k value the more the underfit the model, while providing what the variance shows is the most consistent model. It is a good balance between minimizing the error of the model with the most consistency and providing a generalization of the data points.

In real-world applications, I would be moderately confident in my choice of k = 3. I am confident in that while still a smaller number of neighbors to use, it reduces the model's sensitivity to noise. For example, if there was a bad/incorrect data point in the data set and k = 1, then a predicted point would be classified the same as that bad data point. A k value of 3 allows for more than just one data point to be looked at for classification. What makes me less confident is that not all data is as clean as this data set was. Larger k values are typically needed when data sets have a lot of noise and/or outliers, which most likely most raw data in the world has. If this was to be used on scaled, relevant feature, and outlier processed data sets in the real-world, I would be more confident in my choice.

Findings/Conclusion

After the cross validation error testing, it was found that the linear regression model ended up producing the smallest average testing error, which was 0.01144928, and the smallest testing error variance, which was 2.365601e-05. If I were to pick a model to use for this data set to classify a 2 or 7, I would choose the linear regression model due to it having these results. If I were to pick the KNN model to use, I would select k=3 as my model's k value. Both models resulted in better error values after cross validation happened as opposed to being used on the testing data set which I found surprising. I originally would have thought without the cross validation the error numbers would be better because it was two unchanging data sets. I then remembered that the Monte Carlo technique adds bias due to some data being able to be used multiple times in the testing data set. I would be interested to see what the numbers would have been if k-fold cross validation was used instead, or if my thought process was just incorrect. I was also surprised to see how overfit the k = 1 training error was. It was interesting to see that it said that the model perfectly fit the data set with an error value of 0. That alone shows all the reason for using testing data and cross validation to further test a model that looks too good to be true.

Appendix

```
## 1. Read Training data
ziptrain <- read.table(file= "zip.train.csv", sep = ",");
ziptrain27  <- subset(ziptrain, ziptrain[,1]==2 | ziptrain[,1]==7);
## some sample Exploratory Data Analysis
dim(ziptrain27);    ## 1376 257
sum(ziptrain27[,1] == 2);
sum(ziptrain27[,1] == 7);
barplot(table(ziptrain27$V1), main="Distribution of Digits in Data Set", col = 'purple')
summary(ziptrain27);
cor_matrix <- round(cor(ziptrain27[, -1]),2);
cor_matrix
install.packages("ggcorrplot")
library(ggcorrplot)
# Visualize with ggcorrplot
ggcorrplot(cor_matrix)
## 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(0:32/32),axes=FALSE) ## Also try "col=gray(0:32/32)"
### 2. Build Classification Rules
### linear Regression
mod1 <- lm( V1 ~ . , data= ziptrain27);
pred1.train <- predict.lm(mod1, ziptrain27[,-1]);
pred1.train
```

```
y1pred.train <- 2 + 5*(pred1.train >= 4.5);
## Note that we predict Y1 to $2$ and $7$,
##   depending on the indicator variable whether pred1.train >= 4.5 = (2+7)/2.
train_error_lm <- mean( y1pred.train  != ziptrain27[,1]);
train_error_lm
## KNN
library(class)
# Initialize a vector to store training errors for each k
k_values <- c(1, 3, 5, 7, 9, 11, 13, 15)
train_errors_knn <- numeric(length(k_values))

# Compute training errors for each k
for (i in seq_along(k_values)) {
  k <- k_values[i]
  ypred_knn <- knn(ziptrain27[, -1], ziptrain27[, -1], ziptrain27[, 1], k = k)
  train_errors_knn[i] <- mean(ypred_knn != ziptrain27[, 1])
}

# Print training errors for each k
train_results <- data.frame(k = k_values, Training_Error = train_errors_knn)
print(train_results)

### 3. 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
## Testing error of KNN, and you can change the k values.
xnew2 <- ziptest27[,-1];       ## xnew2 is the X variables of the  "testing" data
ytest <- ziptest27[, 1]   # Response variable

# Predict on the testing data using the linear regression model
pred1.test <- predict(mod1, ziptest27)

# Convert continuous predictions to discrete classes (2 or 7)
y1pred.test <- 2 + 5 * (pred1.test >= 4.5)

# Calculate testing error
test_error_lm <- mean(y1pred.test != ytest)
test_error_lm

# Initialize a vector to store testing errors for each k
test_errors_knn <- numeric(length(k_values))

# Compute testing errors for each k
for (i in seq_along(k_values)) {
  k <- k_values[i]
  ypred_knn_test <- knn(ziptrain27[, -1], xnew2, ziptrain27[, 1], k = k)
  test_errors_knn[i] <- mean(ypred_knn_test != ytest)
}

# Print testing errors for each k
```

```
test_results <- data.frame(k = k_values, Testing_Error = test_errors_knn)
test_results

### 4. Cross-Validation
### The following R code might be useful, but you need to modify it.
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));
  zip27traintemp <- zip27full[flag,];  ## temp training set for CV
  zip27testtemp  <- zip27full[-flag,]; ## temp testing set for CV
  ### you need to write your own R code here to first fit each model to "zip27traintemp"
  ### then get the testing error (TE) values on the testing data "zip27testtemp"

  mod <- lm(V1 ~ ., data = zip27traintemp)  # Fit linear regression
  pred <- predict(mod, zip27testtemp[,-1]) # Predict on the testing set
  te0 <- mean((2 + 5 * (pred >= 4.5)) != zip27testtemp[, 1]) # Testing error
```

```
 te1 <- mean(knn(zip27traintemp[,-1], zip27testtemp[,-1], zip27traintemp[, 1], k = 1) !=
zip27testtemp[, 1])

 te2 <- mean(knn(zip27traintemp[,-1], zip27testtemp[,-1], zip27traintemp[, 1], k = 3) !=
zip27testtemp[, 1])

 te3 <- mean(knn(zip27traintemp[,-1], zip27testtemp[,-1], zip27traintemp[, 1], k = 5) !=
zip27testtemp[, 1])

 te4 <- mean(knn(zip27traintemp[,-1], zip27testtemp[,-1], zip27traintemp[, 1], k = 7) !=
zip27testtemp[, 1])

 te5 <- mean(knn(zip27traintemp[,-1], zip27testtemp[,-1], zip27traintemp[, 1], k = 9) !=
zip27testtemp[, 1])

 te6 <- mean(knn(zip27traintemp[,-1], zip27testtemp[,-1], zip27traintemp[, 1], k = 11) !=
zip27testtemp[, 1])

 te7 <- mean(knn(zip27traintemp[,-1], zip27testtemp[,-1], zip27traintemp[, 1], k = 13) !=
zip27testtemp[, 1])

 te8 <- mean(knn(zip27traintemp[,-1], zip27testtemp[,-1], zip27traintemp[, 1], k = 15) !=
zip27testtemp[, 1])


 ### IMPORTANT: when copying your codes in (2) and (3), please change to

 ###     these temp datasets, "zip27traintemp" and "zip27testtemp" !!!

 ###

 ### Suppose you save the TE values for these 9 methods (1 linear regression and 8 KNN)
as

 ###  te0, te1, te2, te3, te4, te5, te6, te7, te8 respectively, within this loop

 ###   Then you can save these $9$ Testing Error values by using the R code

 ###   Note that the code is not necessary the most efficient

 TEALL = rbind( TEALL, cbind(te0, te1, te2, te3, te4, te5, te6, te7, te8) );

}

TEALL

### Of course, you can also get the training errors if you want
```

```
dim(TEALL);  ### This should be a Bx9 matrices

### if you want, you can change the column name of TEALL

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 ###
```