Elliott Sperin

Handwriting Prediction Analysis

Problem Statement: Using data collected from scanned handwritten numbers collected by the postal service, create a regression and KNN model that can effectively predict digits based on scanned handwritten.

Data description: As described by the dataset source: “Normalized handwritten digits, automatically scanned from envelopes by the U.S. Postal Service. The original scanned digits are binary and of different sizes and orientations; the images here have been deslanted and size normalized, resulting in 16 x 16 grayscale images (Le Cun et al., 1990). " The dataset is a measurement of 256 greyscale values used to predict handwritten numbers 0 through 9. The greyscale values are likely generated using a neural network algorithm that placed the best weight to generate the appropriate number. This dataset is made available by the AT&T research labs.

Exploratory Data Analysis:

Size of dataset: The training dataset includes 7291 observations and 257 variables. The subset training dataset when the dependent variable is 6 and 5 includes 1220 observations and 257 variables. The test dataset includes 2007 observations and 257 variables and the subset of the test dataset when the dependent variable is 6 and 5 includes 330 observations and 257 variables. For the purposes of our analysis, we will be looking at the data when the dependent variable is 5 & 6. The combined size of the training and test datasets is 257 variables, 9299 observations for the full dataset and 1550 for the subset data set when the dependent variable is 6 and 5.

The testing dataset is 27.5% the size of the training dataset. Both datasets are sizable representation of the data.

Variable Descriptions: The dependent variable is a predicted number, 0 through 9 (5 and 6 in our case) by the other variables in the dataset. The other 256 variables are greyscale values collected from neural network algorithm to predict handwritten number values. The greyscale values each hold values between –1 and 1.

Figure 1 visually shows the correlation between the variables. Because of the dark blue color, you can visually see there is not a strong correlation between the independent variables; which makes this dataset useful for analysis. Although it's hard to view specific values in the heatmap, it is still useful to demonstrate the correlation between the 257 variables in the dataset. The heatmap shows trends with different color intensities making it an effective way to compare the correlation between the 257 variables. The lighter color lines running diagonally across the heat map are the intersection between the same variable, which is why it is strongly correlated in these areas.

There are no outliers in the dataset. This is likely because the data is collected by a controlled neural network algorithm that normalized each value in between –1 and 1.

Figure 1: Correlation heat map

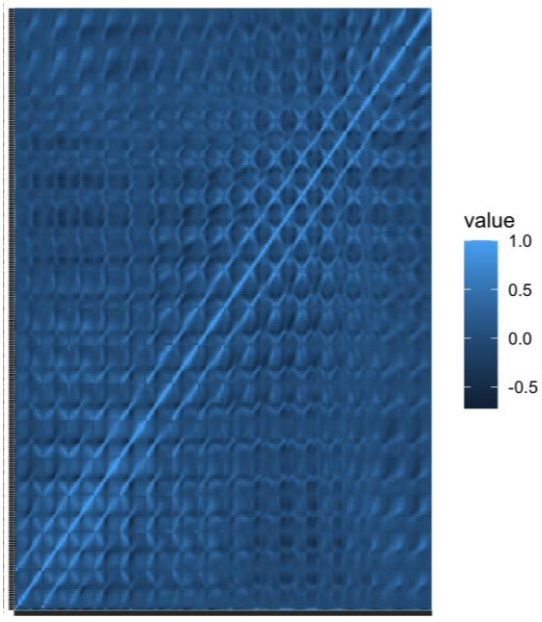
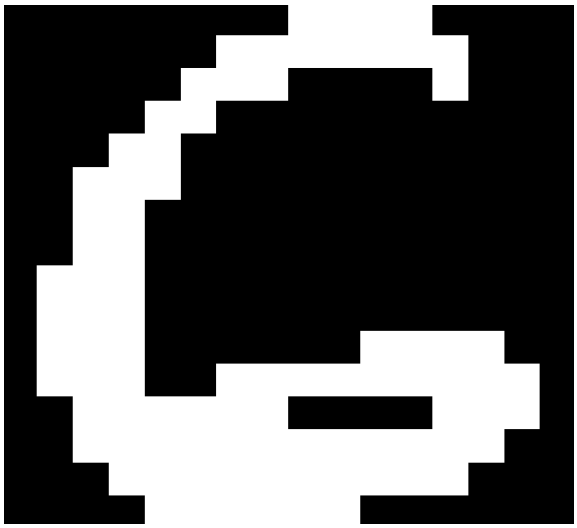


Figure 2: image from data entry



Methods/Methodology:

- We chose to use KNN and Regression models to predict which values are needed to increase the performance to generate an image of a 5 or a 6.

- To give us a starting point we chose the results of at index 10 to see what one of the data entries looks like. *Figure 2* shows us that the generated image from the dataset certainly looks like a six.
- For the regression model, we selected V1 as the response variable and all other variables as the predictor variable, using the Ziptrain56 dataset. We then used a prediction function to predict response values based on the regression model and the ziptrain56 predictor variables. To finish we rounded each response to a binary outcome of either 5 or 6. The average amount the model predicted was different to the actual data was .0049. When using the model on the training data set it performs well, as expected.
- The KNN model requires a k value to be given to determine how many closest values will be used to determine prediction. For our model we chose a k value of 3. Using V1 as the response variable and all other variables as the predictor variables, and a k-nearest-number as 3 we created the KNN model. The average amount the model predicted was different to the actual data, or the bayes classifier, was .0049. The same as the regression model.
- To test error of KNN we created a new model using the ziptest56 dataset. Using a k value of 3, the average amount the model predicted was different to the actual data was 0.0030.
- Cross Validation will help us validate the results of our models. We will perform Monte Carlo Cross Validation. To begin we will combine both the Ziptrain and Ziptest datasets, and then perform cross validation by taking random subset samples of the large dataset that will be split into training and test data. We will generate 100 subsets of the data that will be used to train and test each model to find which model produces the lowest mean testing error and testing error variance. We will perform a cross validation test on a linear regression model and KNN models with k values of 1, 3, 5, 7, 9, 11, 13, and 15.

Results: Cross validation produced the following results:

Mean Testing Error Values for each model.

linearRegression	KNN1	KNN3	KNN5	KNN7	KNN9
0.025939394	0.010272727	0.009242424	0.010969697	0.011454545	0.013212121
KNN11	KNN13	KNN15			
0.014969697	0.016060606	0.016757576			

Variance of Testing Error Values for each model.

linearRegression	KNN1	KNN3	KNN5	KNN7	KNN9
8.224022e-05	2.632292e-05	2.288727e-05	2.704362e-05	3.053121e-05	3.868251e-05
KNN11	KNN13	KNN15			
4.430346e-05	3.979186e-05	4.163861e-05			

Figure 3 visually represents the error amounts for the tested models. The model that performed best during cross validation will be the chosen model. *Figure 4* shows the variance or spread in model results. Note: Index corresponds with the order of the models as listed above.

KNN3 had the best performance as it produced the lowest mean testing error value of .009242424 and had the lowest variance of testing error values of 2.288727e-05. These values measure the amount the model's predicted value is different from the real values and the spread of the model results for the cross-validation iterations. The model predicted the response variable with less than one percent error.

Figure 3: The mean model performance

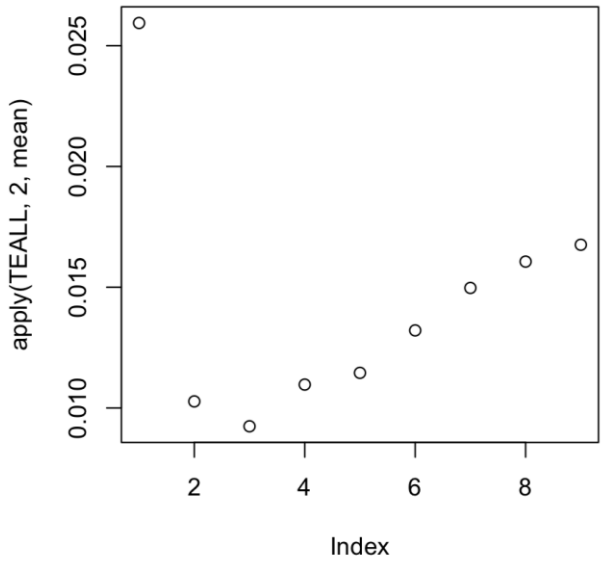
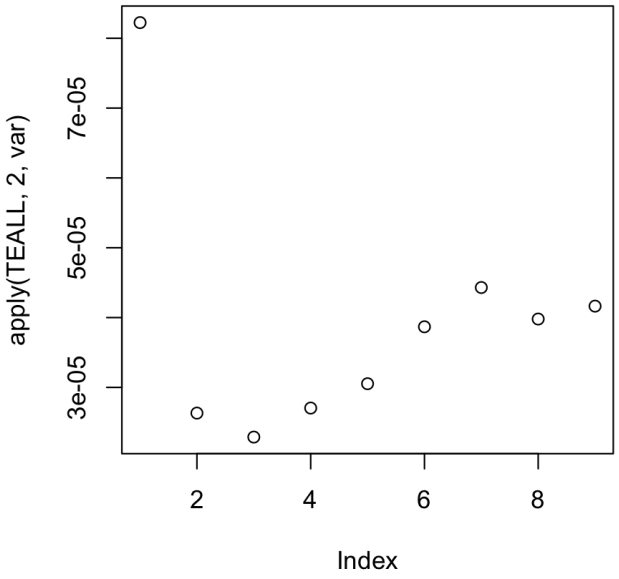


Figure 4: The Variance of model results



Conclusion: The results show that a KNN model when the number of k nearest neighbors is three is the best performing model for this dataset.

The KNN K = 3 model provided decent results when uusing the training dataset than when using Monte Carolo cross validation, so why did we not use the model from the training dataset instead? The reason is because of overfitting data. When using the same dataset to both train and test your data the model picks up the random pattern that might be exclusive to that training dataset. This model would perform very well on the dataset we trained it on, but it decreases in effectiveness when used on new dataset. This can be seen in our analysis. When we tested the model on the training dataset, we had an error of 0.012, but when we use the testing dataset on that same model, we had an error of 0.048. The latter error is a better representation of the model's performance on real data.

The KNN K = 3 model's results when using Monte Carlo Cross Validation .00924. This is a lower error than the model that was tested with the training dataset. This shows the importance of proper validation in classification modeling.

Our analysis demonstrates the importance of cross validation to preserve data and to create the best model.

In our analysis we were able to implement 100% of our data to be trained on and tested. Data collection in the real world is extremely valuable, it is significant to be able to put 100% of a dataset to use in model creation. According to distribution the more data we use the more its reflection the distribution of data in the real world and mitigates random patterns.

In our analysis the best performing model was trained and tested using cross validation. This shows that it would be a costly mistake to finalize a model without using proper cross validation.

Citation:

"Handwritten Digit Recognition with a Back-Propagation Network", Y. Le Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, D. Henderson, *1990 https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf*

Appendix -

```
## Below assume that you save the datasets in the folder ''C://Temp" in your laptop

library(lattice)

library(ggplot2)

library(reshape2)


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

ziptrain56 <- subset(ziptrain, ziptrain[,1]==5 | ziptrain[,1]==6);


## some sample Exploratory Data Analysis

dim(ziptrain56); ## 1220 257
```

```
sum(ziptrain56[,1] == 5); ##556
sum(ziptrain56[,1] == 6); ##664
summary(ziptrain56);
corr_mat <- round(cor(ziptrain56),2);

melted_corr_mat <- melt(corr_mat)
melted_corr_mat

ggplot(data = melted_corr_mat, aes(x=Var1, y=Var2, fill=value))
geom_tile()



## To see the letter picture of the 10-th row by changing the row observation to a matrix
rowindex = 10; ## You can try other "rowindex" values to see other rows
ziptrain56[rowindex, 1];
Xval = t(matrix(data.matrix(ziptrain56[,-1])[rowindex,],byrow=TRUE,16,16)[16:1,]);
image(Xval,col=gray(0:1),axes=FALSE) ## Also try "col=gray(0:32/32)"
### 2. Build Classification Rules
### linear Regression
mod1 <- lm( V1 ~ . , data= ziptrain56);
mod1
pred1.train <- predict.lm(mod1, ziptrain56[,-1]);
y1pred.train <- 5 + (pred1.train >= 5.5);

## Note that we predict Y1 to $5$ and $6$,
## depending on the indicator variable whether pred1.train >= 5.5 = (5+6)/2.
mean( y1pred.train != ziptrain56[,1]);
## KNN
library(class);
kk <- 100; ##KNN with k=3
```

```
xnew <- ziptrain56[,-1];
ypred2.train <- knn(ziptrain56[,-1], xnew, ziptrain56[,1], k=kk);
mean( ypred2.train != ziptrain56[,1])
### 3. Testing Error
### read testing data
ziptest <- read.table(file="zip.test.csv", sep = ",");
ziptest56 <- subset(ziptest, ziptest[,1]==5 | ziptest[,1]==6);
dim(ziptest56) ##330 257
## Testing error of KNN, and you can change the k values.
xnew2 <- ziptest56[,-1]; ## xnew2 is the X variables of the "testing" data
kk <- 3; ## below we use the training data "ziptrain56" to predict xnew2 via KNN
ypred2.test <- knn(ziptrain56[,-1], xnew2, ziptrain56[,1], k=kk);
mean( ypred2.test != ziptest56[,1]) ## Here "ziptest56[,1]" is the Y response of the "testing" data
### 4. Cross-Validation
### The following R code might be useful, but you need to modify it.
zip56full = rbind(ziptrain56, ziptest56) ### combine to a full data set
n1 =dim(ziptrain56)[1]; # training set sample size
n2= dim(ziptest56)[1]; # testing set sample size
n = dim(zip56full)[1]; ## the total sample size
set.seed(7406); ### you can also set othernumber for randomization seed if you want
### 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));
  zip56traintemp <- zip56full[flag,]; ## temp training set for CV
  zip56testtemp <- zip56full[-flag,]; ## temp testing set for CV
  ### you need to write your own R code here to first fit each model to "zip56traintemp"
  ### then get the testing error (TE) values on the testing data "zip56testtemp"
  ###
```

```
#LM
mod1 <- lm( V1 ~ . , data= zip56traintemp);
pred1.train <- predict.lm(mod1, zip56testtemp[,-1]);
y1pred.train <- 5 + (pred1.train >= 5.5);

te0 <- mean( y1pred.train != zip56testtemp[,1]);

#KNN
xnew3 <- zip56testtemp[,-1];
kk <- 1;
ypred2.test <- knn(zip56traintemp[,-1], xnew3, zip56traintemp[,1], k=kk);
te1 <- mean( ypred2.test != zip56testtemp[,1])

xnew3 <- zip56testtemp[,-1];
kk <- 3;
ypred2.test <- knn(zip56traintemp[,-1], xnew3, zip56traintemp[,1], k=kk);
te2 <- mean( ypred2.test != zip56testtemp[,1])

kk <- 5;
ypred2.test <- knn(zip56traintemp[,-1], xnew3, zip56traintemp[,1], k=kk);
te3 <- mean( ypred2.test != zip56testtemp[,1])

kk <- 7;
ypred2.test <- knn(zip56traintemp[,-1], xnew3, zip56traintemp[,1], k=kk);
te4 <- mean( ypred2.test != zip56testtemp[,1])

kk <- 9;
ypred2.test <- knn(zip56traintemp[,-1], xnew3, zip56traintemp[,1], k=kk);
te5 <- mean( ypred2.test != zip56testtemp[,1])
```

```
 kk <- 11;
 ypred2.test <- knn(zip56traintemp[,-1], xnew3, zip56traintemp[,1], k=kk);
 te6 <- mean( ypred2.test != zip56testtemp[,1])
 kk <- 13;
 ypred2.test <- knn(zip56traintemp[,-1], xnew3, zip56traintemp[,1], k=kk);
 te7 <- mean( ypred2.test != zip56testtemp[,1])
 kk <- 15;
 ypred2.test <- knn(zip56traintemp[,-1], xnew3, zip56traintemp[,1], k=kk);
 te8 <- mean( ypred2.test != zip56testtemp[,1])

 ### IMPORTANT: when copying your codes in (2) and (3), please change
 ### the datasets "ziptrain56" and "ziptest56" to
 ### these temp datasets, "zip56traintemp" and "zip56testtemp" !!!
 ### Otherwise you are not doing cross-validations!
 ###
 ### Within each b-th loop, you are given two temp datasets,
 ### you need to write your own codes to use the temp training datae "zip56traintemp"
 ### to fit one of these 9 methods (1 linear regression and 8 KNN);
 ### and then evaluate the performance of these 9 methods
 ### on the temp testing dataset "zip56testtemp" by reporting the correpsonding TE value
 ###
 ### 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) );
 ###
 ### Of course, if you want, you can also report the cross-validation training errors
 ### In many applications, cross-validation testing errors are the most important.
}
```

```
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
plot(apply(TEALL, 2, mean));
apply(TEALL, 2, var);
```