CHARACTER RECOGNITION WITH LOGISTIC REGRESSION

By: HOANG UYEN LE

1. Data cleaning and visualizing the dataset.

The MNIST datasets contain gray-scale images of hand-drawn digits from zero to nine. Each image is 28 pixels in height and 28 pixels in width, so the total is 784 pixels that encoded as each row in the dataset. Each pixel has a single associated pixel-value, indicating the lightness or darkness of that pixel, higher numbers meaning darker. This pixel-value is an integer and ranges between 0 and 255.

The training dataset has 785 columns with the first column called "label", which is the actual digit that was drawn by user, and the rest of the columns contain the pixel-values of the associated image. There are 59999 samples of hand-written digits in the dataset.

I plotted some images to see how the digits are drawn

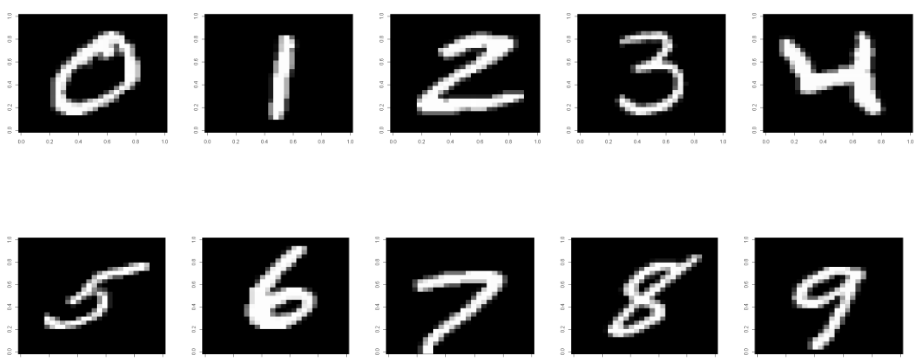

Figure 1: 0 – 9 hand-drawn digits visualized from train dataset

Then, to see the quantity of unique values appear in the training dataset, I plotted bar plot with attached command. "One" has the highest count, nearly 7000 times in the training sample, while others shared the similar quantity at around 6000 (***Figure 2***)

```
unique(train$label)
digitTable <- table(train$label)
digitTable
class(train$label)

ggplot(train, aes(x = factor(label), fill = label)) +
                geom_bar() +
                xlab("Digits") +
                ylab("Digit Count") +
                ggtitle("Total Number of Digits in Training Dataset")
```

Figure 2: Total number of Digits in Training dataset

2. Building the model

Ten different logistic regression models are made, one for each possible digit 0-9. Instead of having a 'label' indicating which number a particular sample is in train dataset, I created new sub training set for each digit and now the "label" columns in these sub training sets are binary with a 1 if it corresponds to the digit that is trained for, and 0 otherwise.

```
### Classify each digit
zero <- as.numeric(train$label == 0)
one <- as.numeric(train$label == 1)
two <- as.numeric(train$label == 2)
three <- as.numeric(train$label == 3)
four <- as.numeric(train$label == 4)
five <- as.numeric(train$label == 5)
six <- as.numeric(train$label == 6)
seven <- as.numeric(train$label == 7)
eight <- as.numeric(train$label == 8)
nine <- as.numeric(train$label == 9)
```

```
### Create sub dataframe for each digit
trainzero <- cbind(zero, train[,-1])
trainone <- cbind(one, train[,-1])
traintwo <- cbind(two, train[,-1])
trainthree <- cbind(three, train[,-1])
trainfour <- cbind(four, train[,-1])
trainfive <- cbind(five, train[,-1])
trainsix <- cbind(six, train[,-1])
trainseven <- cbind(seven, train[,-1])
traineight <- cbind(eight, train[,-1])
trainnine <- cbind(nine, train[,-1])
```

Using 10 subsets to build the generalized linear model to predict the label as a function of the image pixels for each digit. Generalized Linear Models are an extension of linear regression models that allow the dependent variable to be non-normal. If the condition mean $E(y|x) = \mu$ cannot be directly expressed as a linear function of the regression parameters, a nonlinear transformation $\Theta\,(E(y|x)) = \Theta(\mu) = log(\mu) = \sum_i \beta_i x_i$ can be linear.

glm() command was called to run a logistic regression, regressing label on image pixels. I specified that the distribution is binomial and logit link, so the estimates are in logits. We can revert these logits to odds ratios μ with $\mu = \Theta^{-1}(\Theta(\mu))$ to see the influence of pixel values on dependent variable "label".

```
### Build the models
probzero <- glm(zero ~ . , data = trainzero, family = binomial(link="logit"))
probone <- glm(one ~ . , data = trainone, family = binomial(link="logit"))
probtwo <- glm(two ~ . , data = traintwo, family = binomial(link="logit"))
probthree <- glm(three ~ . , data = trainthree, family = binomial(link="logit"))
probfour <- glm(four ~ . , data = trainfour, family = binomial(link="logit"))
probfive <- glm(five ~ . , data = trainfive, family = binomial(link="logit"))
probsix <- glm(six ~ . , data = trainsix, family = binomial(link="logit"))
probseven <- glm(seven ~ . , data = trainseven, family = binomial(link="logit"))
probeight <- glm(eight ~ . , data = traineight, family = binomial(link="logit"))
probnine <- glm(nine ~ . , data = trainnine, family = binomial(link="logit"))
```

```
### Convert logits to odd ratios
exp(coef(probzero))
exp(coef(probone))
exp(coef(probtwo))
exp(coef(probthree))
exp(coef(probfour))
exp(coef(probfive))
exp(coef(probsix))
exp(coef(probseven))
exp(coef(probeight))
exp(coef(probnine))
```

Original testing dataset shares the same structures as training, with 9999 observations. Keeping "label" column later for comparison and accuracy stage, and only using image pixel values to apply into 10 built logistic models. Likelihood probabilities regarding 0-9 digits of each observation are assembled to new data frame deliberately with numeric order.

```
### Applying the models
ProbabilityOfEachValue <- data.frame(predict(probone, newdata = test, type = “response” ),
                    predict(probtwo, newdata = test, type = “response”),
                    predict(probthree, newdata = test, type = “response”),
                    predict(probfour, newdata = test, type = “response”),
                    predict(probfive, newdata = test, type = “response”),
                    predict(probsix, newdata = test, type = “response”),
                    predict(probseven, newdata = test, type = “response”),
                    predict(probeight, newdata = test, type = “response”),
                    predict(probnine, newdata = test, type = “response”),
                    predict(probzero, newdata = test, type = “response”))
colnames(ProbabilityOfEachValue) <- c("One","Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Zero")
```

The output of particular probabilities kind of unrelated, and softmax function has to be implemented. Softmax function will normalize these values into a probability distribution whose total sums up to 1 based on this equation

$$P(y=j \mid \Theta^{(i)}) = \frac{e^{\Theta^{(i)}}}{\sum_{j=0}^{k} e^{\Theta_k^{(i)}}}$$

Softmax function

where $\Theta = w_0x_0 + w_1x_1 + \ldots + w_kx_k = \sum_{i=0}^{k} w_ix_i = w^Tx$

over the ten digits, and the predicted digit is the one associated with the maximum probability.

```
### Creating similar-structured dataframe
softmax_dataframe <- data.frame(One = numeric(), Two = numeric(), Three = numeric(), Four = numeric(), Five = numeric(),
                                Six = numeric(), Seven = numeric(), Eight = numeric(), Nine = numeric(), Zero = numeric())
### Softmax function
softmax <- function (x) {
 exp(x - max(x) - log(sum(exp(x - max(x)))))}
### Looping softmax function to each row of Probability dataset and assemling result to new dataframe
convert <- function(data){
 for (i in seq(nrow(data))){
  x = data[i,]
  softmax_dataframe <- rbind(softmax_dataframe, softmax(data[i,]))
 }
 return (softmax_dataframe)
}
### Presenting dataframe after converting by softmax
softmax_dataframe <- convert(ProbabilityOfEachValue)
```

From 10 calculated probabilities, the prediction will be assigned to digit which pertains the highest probability, because it correlates to the model that is the most confident. The 0's are in the 10th index, so any guess that is a 10 is changed to a 0. It's time to query the actual "label" column of test dataset to compare with predicted values. Also executing the confusion matrix to calculate accuracy statistics index.

```
### Find Predicted value by taking the highest probability
Label <- rep(NA, nrow(softmax_dataframe))
for (i in seq(nrow(softmax_dataframe)))
{ Label[i] <- which.max(softmax_dataframe[i,])}
Label[Label == 10] <- 0
Label <- as.data.frame(Label)
### Create reference table for comparison
read <- read.csv("mnist_test.csv")
read <- read[1]
reference <- cbind(Label,read)
colnames(reference) <- c("Prediction", "Actual")
reference$Prediction <- as.factor(reference$Prediction)
reference$Actual <- as.factor(reference$Actual)
### Confusion matrix
confusionMatrix(reference$Prediction, reference$Actual)
```

Table 1 is the comparison matric of actual and predicted values of 9999 observation in testing dataset, and 6658 out of 9999 values are right predicted, so the accuracy rate of this model is approximately 66.6% computed along with a 95% CI (from 0.6565 to 0.6721). Some other statistical indexes provided by confusion matric command, such as p-value of this model is < 2.2e-16; No information rate is around 20.31%. Also, class for digit 8 has the lowest Sensitivity and Balanced accuracy rates, while it has up to 1000 wrong predicted values.

		Prediction									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Actual	**0**	**382**	0	35	42	3	10	96	2	407	3
	1	0	**1114**	8	3	0	1	5	1	3	0
	2	1	18	**955**	16	9	0	8	4	20	1
	3	0	2	108	**889**	0	1	0	2	8	0
	4	2	4	18	64	**874**	0	4	0	15	1
	5	2	5	24	319	43	**432**	15	5	45	2
	6	1	6	103	22	35	9	**782**	0	0	0
	7	2	18	92	244	27	3	1	**625**	13	2
	8	1	14	47	258	28	15	41	5	**565**	0
	9	2	10	5	174	158	1	6	15	598	**40**

Table 1: Confusion matrix of Actual vs. Prediction digits

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2    3    4    5    6    7    8    9
         0  382    0   35   42    3   10   96    2  407    3
         1    0 1114    8    3    0    1    5    1    3    0
         2    1   18  955   16    9    0    8    4   20    1
         3    0    2  108  889    0    1    0    2    8    0
         4    2    4   18   64  874    0    4    0   15    1
         5    2    5   24  319   43  432   15    5   45    2
         6    1    6  103   22   35    9  782    0    0    0
         7    2   18   92  244   27    3    1  625   13    2
         8    1   14   47  258   28   15   41    5  565    0
         9    2   10    5  174  158    1    6   15  598   40

Overall Statistics

               Accuracy : 0.6659
                 95% CI : (0.6565, 0.6751)
    No Information Rate : 0.2031
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6284

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity           0.97201   0.9353  0.68459  0.43772  0.74257  0.91525  0.81628  0.94841  0.33751  0.81633
Specificity           0.93775   0.9976  0.99105  0.98481  0.98776  0.95172  0.98053  0.95696  0.95087  0.90261
Pos Pred Value        0.38980   0.9815  0.92539  0.88020  0.89002  0.48430  0.81628  0.60857  0.58008  0.03964
Neg Pred Value        0.99878   0.9913  0.95093  0.87296  0.96640  0.99561  0.98053  0.99621  0.87712  0.99900
Prevalence            0.03930   0.1191  0.13951  0.20312  0.11771  0.04720  0.09581  0.06591  0.16742  0.00490
Detection Rate        0.03820   0.1114  0.09551  0.08891  0.08741  0.04320  0.07821  0.06251  0.05651  0.00400
Detection Prevalence  0.09801   0.1135  0.10321  0.10101  0.09821  0.08921  0.09581  0.10271  0.09741  0.10091
Balanced Accuracy     0.95488   0.9665  0.83782  0.71126  0.86516  0.93349  0.89841  0.95268  0.64419  0.85947
```

Reference:

- Matt, B. (April 17, 2017) – A One – Stop Shop for Principal Component Analysis

Retrieved from: https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

- Kyle S. (June 25, 2017) – Digit Recognition 1

Retrieved from: https://rpubs.com/kstahl/MNIST-1

- Hamza, M. (Nov 26, 2018) – The Softmax Function, Simplified

Retrieved from: https://towardsdatascience.com/softmax-function-simplified-714068bf8156

- Tyler, M. – Principal Component Analysis of Handwritten Digits

Retrieved from: https://static1.squarespace.com/static/55133727e4b05ad39f9b9749/t/56ba9b20a3360ce1d09cb271/1455069985248/principal-component-analysis.pdf