Image Processing with Hiragana Characters

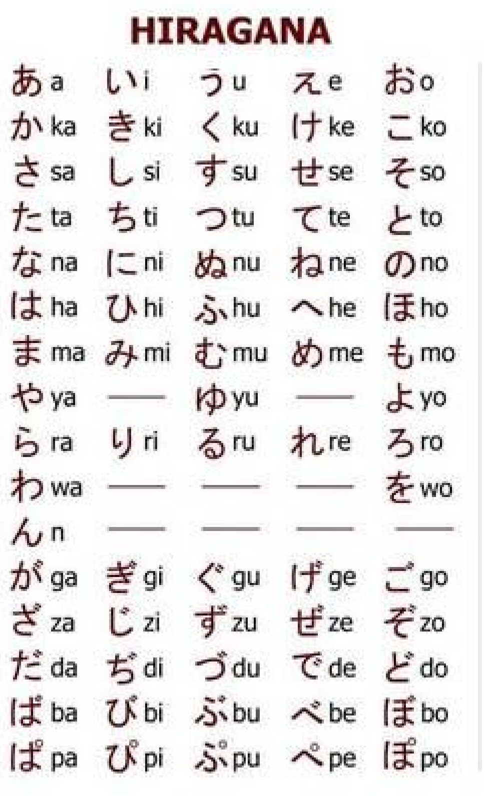
Andy Haas

January 30, 2019

Introduction

The goal of this project is to gather images of many hand-written Japanese characters and train an algorithm to guess the correct Japanese Character. This is very similar to the MNIST digit reader data set. A major problem we are faced with is that there is not a readily available database of handwritten Japanese characters. Simply getting the data prepared and ready for analyis was a great undertaking that took most of the time that went into making this project. The second major problem is that there are many different characters, making it more difficult for an algorithm to predict the correct character. In the MNIST dataset, there are 10 different factors, 0-9. The Japanese hiragana alphabet has more than 70 characters, resulting in a total of 74 factors.

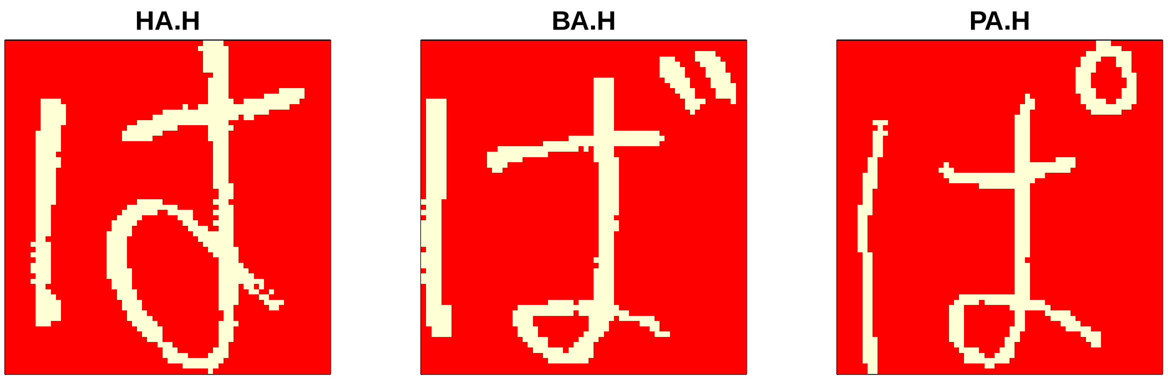
There are thousands and thousands of Japanese kanji characters, but we will only be looking at the basic Japanese alphabet, called 'hiragana'.



HIRAGANA

あ a	い i	う u	え e	お o
か ka	き ki	く ku	け ke	こ ko
さ sa	し si	す su	せ se	そ so
た ta	ち ti	つ tu	て te	と to
な na	に ni	ぬ nu	ね ne	の no
は ha	ひ hi	ふ hu	へ he	ほ ho
ま ma	み mi	む mu	め me	も mo
や ya	—	ゆ yu	—	よ yo
ら ra	り ri	る ru	れ re	ろ ro
わ wa	—	—	—	を wo
ん n	—	—	—	—
が ga	ぎ gi	ぐ gu	げ ge	ご go
ざ za	じ zi	ず zu	ぜ ze	ぞ zo
だ da	ぢ di	づ du	で de	ど do
ば ba	び bi	ぶ bu	べ be	ぼ bo
ぱ pa	ぴ pi	ぷ pu	ぺ pe	ぽ po

With many characters, the only difference between them is a small circle or two dashes in the top right corner. You can see this here. This is “ha”, “ba”, and “pa”.



The data we will be using was gathered by the National Institute of Advanced Industrial Science and Technology (AIST)) under cooperation with the Japan Electronics and Information Technology Industries Association. The data was collected between 1973 and 1984, and was made available for public download in 2011. It is only to be used for academic purposes, and you must apply for permission to download it.

Here is a link to the website. http://etlcdb.db.aist.go.jp/

There are 9 different zipped files with thousands and thousands of records. Finding the record that included the data I was looking for took a good deal of investigation, as I was unfamiliar with their encoding style. All of the data is just a string of bytes that you must read and decode based on the provided specifications. You can find the specifications here. http://etlcdb.db.aist.go.jp/?page_id=2461

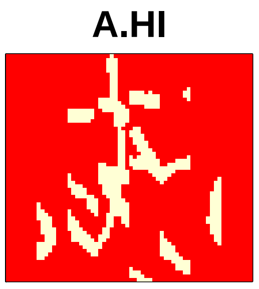
After sifting through the different files, I found that ETL-8B2 includes the hiragana characters I was looking for, and in a format similar to what we see in the MNIST digit reader data set. The image data is represented by zeros and ones.

The ETL-8B2C1 file contains data for 360 different Japanese characters, each written by 160 different people. Each record is 512 bytes. To wrangle the data, we must read the file 512 bytes at a time, and decode the bytes into data we can use.

Lets start by reading the first record in the data and see what we have. The 2 main things we need is the name of the character, and an array of values representing the pixels of the image. Here is the code used to get the first record.

```
# open the file as readable byte 'rb'
my.file <- file('ETL8B2C1', 'rb')
# the first record is a buffer, and empty, so we wil skip the first 512 bytes
# and go to the next 512 bytes
d <- readBin(my.file, raw(), n = 512L)
d <- readBin(my.file, raw(), n = 512L)
close(my.file)
# get name of hiragana character
lab <- rawToChar(d[5:8])
# get image data
img <- as.integer(rawToBits( d[9:512]))
lab
```

```
## [1] "A.HI"
```



The "A.HI" means it is 'A' from the 'hiragana' alphabet. When looking at the image, we see that it is very scrambled, but upon close inspection, we can see it somewhat resembles a Japanese 'A', but something is very wrong. This particular issue caused me quite a bit of trouble.

To understand what went wrong, we first need to understand how the image is encoded. Eeach image is 64 x 63 pixels and 504 bytes. Eech byte must be changed to binary. One byte can be represented by 8 binary digits. 504 bytes x 8 digits = 4,032. 64 x 63 also equals 4,032. So we change all bytes to binary and then turn this binary array of length 4032 into a 64 x 63 matrix, then show the image of the matrix. Here is an example of changing a byte to binary.

```
a <- as.raw(5)
a
```

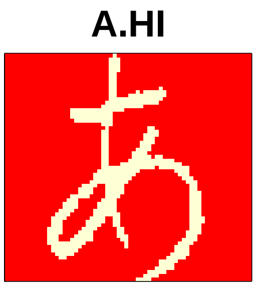
```
## [1] 05
```

```
b <- rawToBits(a)
b
```

```
## [1] 01 00 01 00 00 00 00 00
```

As it turns out, when coercing raw data to binary using the rawToBits() function, R puts the most significant digit on the right; however, this is not how the data was meant to be read. The most significant digit needs be on the left. This causes every 8 pixels to be flipped. If we reverse the order of every 8 binary digits, the image comes out as intended. We do this using sapply and the rev() function.

```r
my.file <- file('ETL8B2C1', 'rb')
d <- readBin(my.file, raw(), n = 512L)
d <- readBin(my.file, raw(), n = 512L)
close(my.file)
lab <- rawToChar(d[5:8])
brev <- function(x) rev(rawToBits(x))
img <- as.integer(paste(sapply(d[9:512],brev)))
```



The next step is to read the whole file and save it to data we can use for machine learning. Here is the code.

```r
#CREATES THE 'HIRAGANA' DATAFRAME

library(tidyverse)

#opens file connection
my.file <- file('ETL8B2C1', 'rb')

#create elements
labels <- c()
image <- list()

# reads the first 512 bites, which are buffer
d <- readBin(my.file, raw(), n = 512L)

#reads 512 bytes at a time, 1 to 322 entries
for (i in 1:12000){
d <- readBin(my.file, raw(), n = 512L)

# read ascii character
labels[i] <- rawToChar(d[5:8])

# read image data
# reverse bytes
brev <- function(x) rev(rawToBits(x))
image[[i]] <- as.integer(paste(sapply(d[9:512],brev)))
}
close(my.file)
```

```r
labels <- as.factor(labels)
image <- matrix(unlist(image), ncol = 4032, byrow = TRUE)
```

Let's look at the structure of our data.

```r
str(labels)
```

```
##  chr [1:12000] "A.HI" "A.HI" "A.HI" "A.HI" "A.HI" "A.HI" "A.HI" "A.HI" ...
```

```r
str(image)
```

```
##  int [1:12000, 1:4032] 0 0 0 0 0 0 0 0 0 0 ...
```

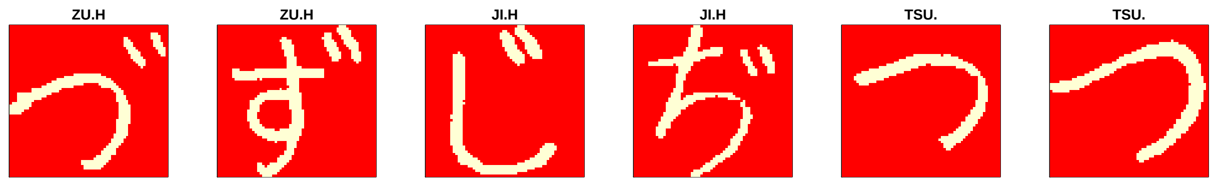
It is not a data frame. Instead, there are 2 objects, 'labels' and 'images'. The object, 'labels', is an array of 72 unique character strings. It is in the form of characters because it still needs small adjustments before being converted to factors. We will get to that later. The obect, 'image', is a large 12000 x 4032 matrix. Each row represents one image. If we want to view the image, we can reshape the row into a 64 x 63 matrix.

Let's check that we actually have 160 entries for each character.

```r
table(labels)
```

```
## labels
## A.HI BA.H BE.H BI.H BO.H BU.H CHI. DA.H DE.H DO.H E.HI FU.H GA.H GE.H GI.H
##  160  160  160  160  160  160  160  160  160  160  160  160  160  160  160
## GO.H GU.H HA.H HE.H HI.H HO.H I.HI JI.H KA.H KE.H KI.H KO.H KU.H MA.H ME.H
##  160  160  160  160  160  160  160  320  160  160  160  160  160  160  160
## MI.H MO.H MU.H N.HI NA.H NE.H NI.H NO.H NU.H O.HI O.WO PA.H PE.H PI.H PO.H
##  160  160  160  160  160  160  160  160  160  160  160  160  160  160  160
## PU.H RA.H RE.H RI.H RO.H RU.H SA.H SE.H SHI. SO.H SU.H TA.H TE.H TO.H TSU.
##  160  160  160  160  160  160  160  160  160  160  160  160  160  160  320
## U.HI WA.H YA.H YA.S YO.H YO.S YU.H YU.S ZA.H ZE.H ZO.H ZU.H
##  160  160  160  160  160  160  160  160  160  160  160  320
```

We notice that the labels "ZU.H", "JI.H", and "TSU." have 320 entries. This is because there are two different ways to write these characters.



We will need to split each of these in half. "ZU.H" will become "ZU.H" and "SUZU". "JI.H" will become "JI.H" and "CHIJI". "TSU" can actually stay the same because the ony difference is size, but the images have been normalized so even a small 'TSU' will appear the same as its larger counterpart.

```r
labels[3201:3360] <- "SUZU"
labels[4481:4640] <- "CHIJI"
labels <- as.factor(labels)
```

Methods

The first step in creating and testing a model is to partition our data into train and test sets. We will randomly seperate using the sample() function, 90% train and 10% test.

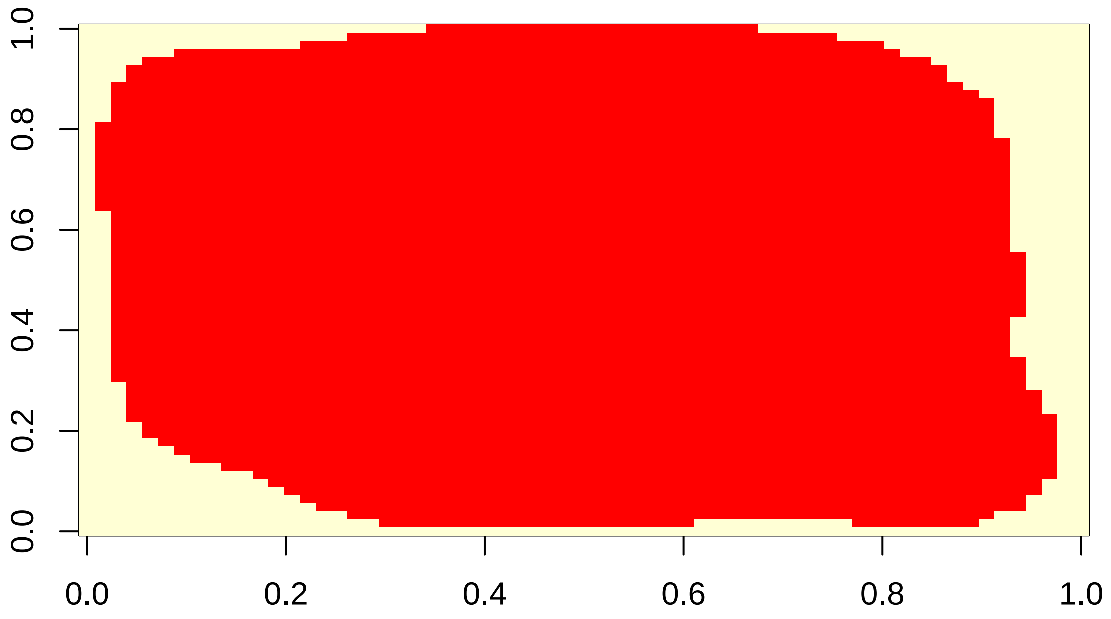
```r
set.seed(1983)
index <- sample(12000, 1200)

y_train<-factor(labels[-index])
x_train <- image[-index,]

y_test<-factor(labels[index])
x_test <- image[index,]
```

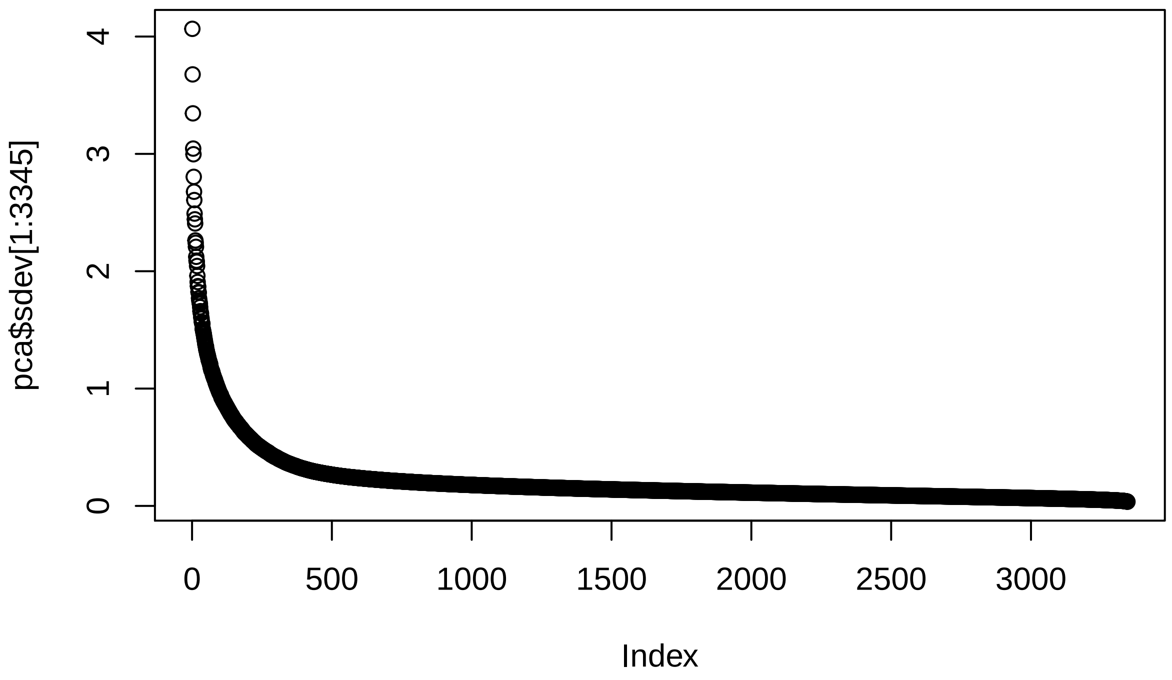
I was hoping to apply numerous machine learning methods to this data set, and finally make an ensemble. After attempting a few, I discovered that the run times were very long, and often didn't finish. To try to alleviate this issue, we will attempt some dimension reduction.

First, We will use the nearZeroVar() function from the caret package. This function identifies the columns that have very low variation. These columns will not help us differentiate between hiragana characters, so they should be ignored. Let's look at which pixels we keep (red), and which we ignore.



We keep 3345 columns of our 4032 original columns.

Next, we calculate the principal components using the prcomp() function to identify the columns that describe the most amount of variability in the data. The runtime on this can be quite long, up to 10 minutes.



This plot shows us principal components in order of most variability to least variability. Notice there are only about 20 columns that have variance above 2. Everything else is very low in comparison.

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
Standard deviation	4.065682	3.677503	3.345613	3.04524	2.997021	2.804504	2.677364	2.606373
Proportion of Variance	0.037990	0.031080	0.025730	0.02131	0.020640	0.018080	0.016480	0.015610
Cumulative Proportion	0.037990	0.069080	0.094800	0.11612	0.136760	0.154840	0.171310	0.186930

This table is quite useful as it shows the proportion of the data that can be described by each principal component. When using this method of dimension reduction on the MNIST data, we noticed that the first 36 columns described more than 80% of the data. Sadly, we do not get such results here. Even with all the PCs from PC1 to PC100, we get a cumulative proportion of about 65%.

	PC95	PC96	PC97	PC98	PC99	PC100	PC101
Standard deviation	0.9831678	0.9782654	0.9742986	0.965599	0.9611017	0.9574677	0.9527301
Proportion of Variance	0.0022200	0.0022000	0.0021800	0.002140	0.0021200	0.0021100	0.0020900
Cumulative Proportion	0.6305400	0.6327400	0.6349200	0.637070	0.6391900	0.6413000	0.6433800

The last thing to do is train the algorithm and test it.

```r
K <- 30
x_train_pca <- pca$x[,1:K]
fit <- knn3(x_train_pca, y_train)
#transform the test set
```

```
x_test_pca <- sweep(x_test[,col_index], 2, col_means) %*% pca$rotation
x_test_pca <- x_test_pca[,1:K]
# make predictions and see results
y_hat <- predict(fit, x_test_pca, type = "class")
results <- confusionMatrix(y_hat, factor(y_test))
results$overall[1]
```

```
##  Accuracy
## 0.8208333
```

Results

When using almost the exact same method on the MNIST data set, we got above 0.97 accuracy. Here, we get an accuracy of about 0.82. Let's look at the Sensitivity and Specificity to see where the mistakes are being made.

```
which(results$byClass[,1]<.6)
```

```
## Class: BA.H Class: BI.H Class: BO.H Class: PI.H Class: PU.H Class: YA.S
##           2           4           5          45          47          66
```

These are the factors that received a sensitivity less than 0.6, These characters are the ones that are identified only by two dashes or a circle in the top-right corner (excluding 'WA'). It seems this is a major issue for this algorithm.

One possible solution is to use a second algorithm that focuses only on the rop-right part of each image to try to identify if it has two dashes, a circle, or neither. This is an extra challenge that I am not prepared to undertake at this time.

Conclusion

Machine learning can be used to identify handwritten Japanese hiragana characters. Here, we used the KNN method. We did not achieve an accuracy as high as we did in the MNIST digit reader data set, but we still did pretty well, achieving an accuracy of 0.82. The issue that caused most of the mistakes was due to many characters looking identical, except for one mark in the corner.

The major difficulty in this project was wrangling the data. Reading raw data and converting it into usable objects took a lot of experimentation. It took weeks to understand the encoding and write code that could convert it into data in the form we needed.

A second issue was in run times. Many of the methods in the caret package are not usable on this data set due to the high number of columns. I feel like with a lot of tinkering, one could overcome this issue through dimension reduction, but I am more than satisfied with what we have accomplished already.

Thank you for taking the time to read this report. I hope you learned something, I know I did!