Challenge 1: Comparing 3 and 4

Dagmawe Haileslassie, Kylie Landa, Erica Meyers & Seth Mutenda

3/16/2022

Dataset Creation

Your dataset should have in total 1000 randomly selected digits (feel free to use a set.seed command so that your results are reproducible). Your training dataset should have 800 observations and your testing should have 200 observations.

Our Approach

We have seen in class that the MNIST database (Modified National Institute of Standards and Technology database) is a large collection of handwritten digits used by the Machine learning community. The `dslabs` packages has a handy function called `read_mnist` that allows to load this dataset as follows:

```
mnist <- read_mnist("~/Mscs 341 S22/Class/Data")
str(mnist)
```

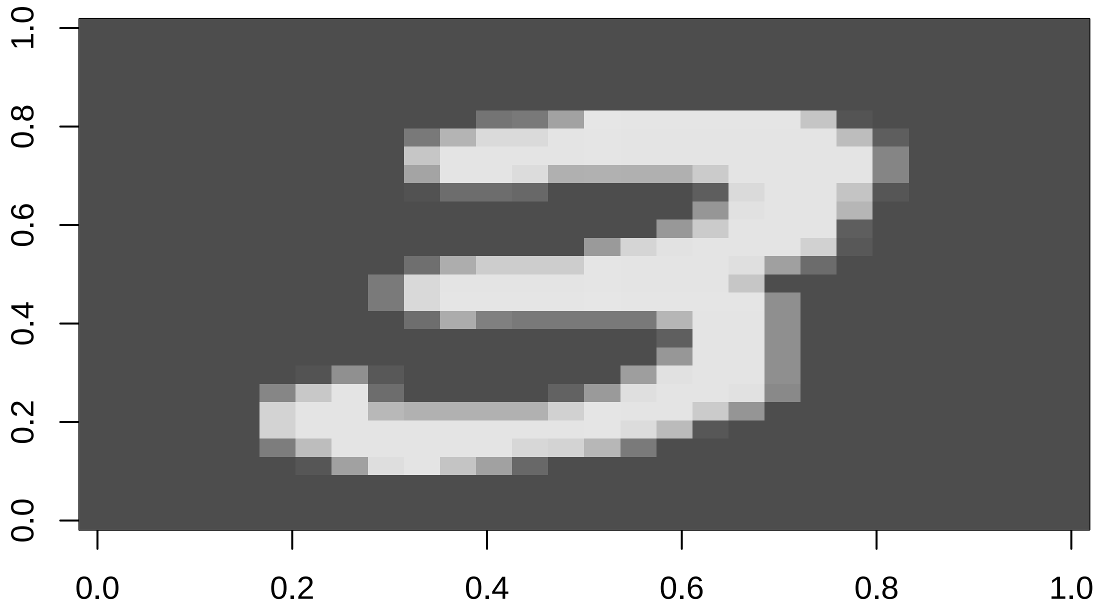
```
## List of 2
##  $ train:List of 2
##   ..$ images: int [1:60000, 1:784] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ labels: int [1:60000] 5 0 4 1 9 2 1 3 1 4 ...
##  $ test :List of 2
##   ..$ images: int [1:10000, 1:784] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ labels: int [1:10000] 7 2 1 0 4 1 4 9 5 9 ...
```

We can see that the Mnist has a training and testing set. The training dataset has 60,000 elements represented as a matrix of 6000×784 (every image is a vector of 784, representing a 28×28 image). It also has the labels corresponding to each of the images represented as integers. Finally the testing dataset has 10,000 elements represented in a similar way.

```
plotImage <- function(dat,size=28){
  imag <- matrix(dat,nrow=size)[,28:1]
  image(imag,col=grey.colors(256), xlab = "", ylab="")
}
```

Let's see an example of a 3 and 4 in our training dataset

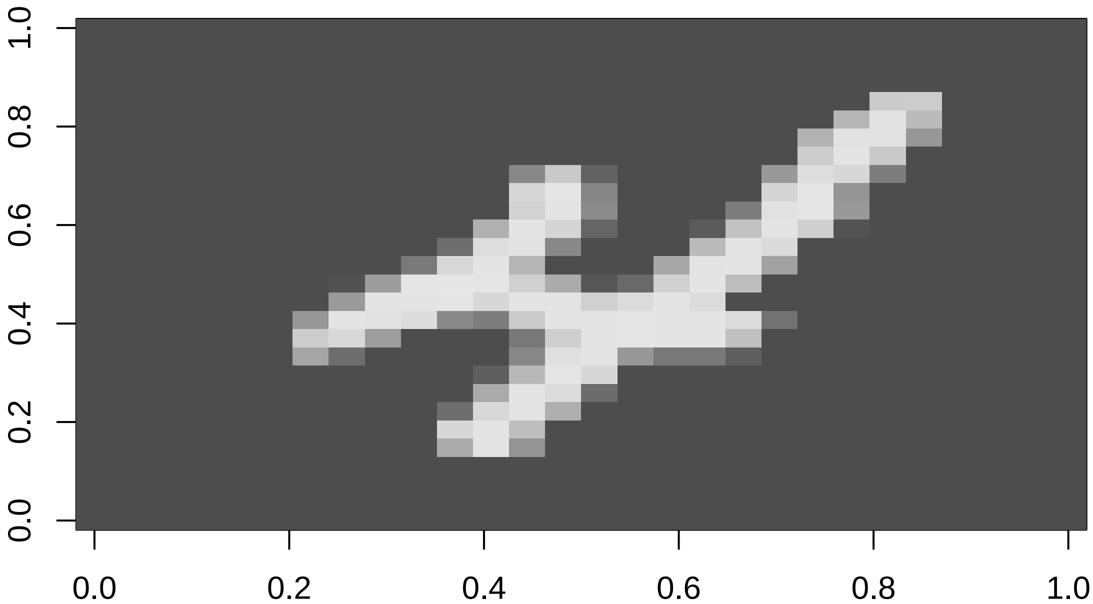
```
plotImage(mnist$train$images[8,])
```




```
mnist$train$labels[8]
```

```
## [1] 3
```

```
plotImage(mnist$train$images[10,])
```




```
mnist$train$labels[10]
```

```
## [1] 4
```

So the problem we are facing now is how do we sift through all of the labels to find numbers that belong to a certain set, the specific set we are looking for is numbers 3 and 4. After finding out how to access these sets we can then look into what exactly makes them different/easily classifiable.

```
#indices for 3
index_of3 <- c()
for (x in 1:length(mnist$train$labels)){
  if(mnist$train$labels[x] == '3'){
    index_of3 <- append(index_of3, x)
  }
}
index_of3 <- index_of3[1:500]

#indices for 4
index_of4 <- c()
for (x in 1:length(mnist$train$labels)){
  if(mnist$train$labels[x] == '4'){
    index_of4 <- append(index_of4, x)
  }
}
index_of4 <- index_of4[1:500]

index_of5 <- c()
```

```r
for (x in 1:length(mnist$train$labels)){
  if(mnist$train$labels[x] == '5'){
    index_of5 <- append(index_of5, x)
  }
}
index_of5 <- index_of5[1:500]

indeces <- tibble(index_of3, index_of4, index_of5)
indeces
```

```
## # A tibble: 500 x 3
##    index_of3 index_of4 index_of5
##        <int>     <int>     <int>
##  1         8         3         1
##  2        11        10        12
##  3        13        21        36
##  4        28        27        48
##  5        31        54        66
##  6        45        59       101
##  7        50        61       133
##  8        51        62       139
##  9        75        65       146
## 10        87        90       174
## # ... with 490 more rows
```

```r
#accessing matrix
accessMatrix <- function(dat,size=28){
  newmatrix <- matrix(dat,nrow=size)[,28:1]
}

#check for number 3
newmatrix3 <- accessMatrix(mnist$train$images[8,])
newmatrix3
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##  [2,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##  [3,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##  [4,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##  [5,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##  [6,]    0    0    0    0   49  208  208   61    0     0     0     0     0
##  [7,]    0    0    0    7  157  252  252  183    5     0     0     0     0
##  [8,]    0    0    0  103  252  252  252  252   75     0     0     0     0
##  [9,]    0    0    0  235  252  252  147   29    9     0     0     0    45
## [10,]    0    0    0  252  252  252  134    0    0     0     0    31   223
## [11,]    0    0    0  172  252  252  134    0    0     0     0   123   253
## [12,]    0    0    0  103  252  252  134    0    0     0     0    52   253
## [13,]    0    0    0   24  217  252  134    0    0     0     0    44   253
## [14,]    0    0    0    0  207  252  203   18    0     0     0    44   253
## [15,]    0    0    0    0  146  253  253   92    0     0     0    44   255
## [16,]    0    0    0    0   45  230  252  239   98     0     0    44   253
## [17,]    0    0    0    0    0  153  252  252  242    86    15   143   253
## [18,]    0    0    0    0    0    8  188  252  252   252   252   252   253
## [19,]    0    0    0    0    0    0   83  243  252   252   252   252   253
## [20,]    0    0    0    0    0    0    0   65   74    74    74    74    74
```

```
## [21,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [22,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [23,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [24,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [25,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [26,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [27,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [28,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##       [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
##  [1,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [2,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [3,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [4,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [5,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [6,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [7,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [8,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [9,]    45     0     0     0     0     0     0     0     0     0     0     0
## [10,]   222    32     0     0     0     4   109   178    43     0     0     0
## [11,]   252   125     0     0     0    29   252   252   139     0     0     0
## [12,]   252   193     0     0     0    29   252   252   224    38     0     0
## [13,]   252   193     0     0     0    24   230   252   226    43     0     0
## [14,]   252   193     0     0     0     0   132   252   252   105     0     0
## [15,]   253   253    91     0     0     0   133   253   253   255     0     0
## [16,]   252   252   212     0     0     0   132   252   252   253     0     0
## [17,]   252   252   247    88     0     0   132   252   252   253     0     0
## [18,]   252   252   252   189    85    14   189   252   252   253     0     0
## [19,]   177   238   252   252   243   226   252   252   252   253     0     0
## [20,]     0   102   252   252   252   252   252   252   252   253     0     0
## [21,]     0    28   204   252   252   252   252   252   252   174     0     0
## [22,]     0     0     9    14   144   172   252   252   158     6     0     0
## [23,]     0     0     0     0     0     7    59    59    14     0     0     0
## [24,]     0     0     0     0     0     0     0     0     0     0     0     0
## [25,]     0     0     0     0     0     0     0     0     0     0     0     0
## [26,]     0     0     0     0     0     0     0     0     0     0     0     0
## [27,]     0     0     0     0     0     0     0     0     0     0     0     0
## [28,]     0     0     0     0     0     0     0     0     0     0     0     0
##       [,26] [,27] [,28]
##  [1,]     0     0     0
##  [2,]     0     0     0
##  [3,]     0     0     0
##  [4,]     0     0     0
##  [5,]     0     0     0
##  [6,]     0     0     0
##  [7,]     0     0     0
##  [8,]     0     0     0
##  [9,]     0     0     0
## [10,]     0     0     0
## [11,]     0     0     0
## [12,]     0     0     0
## [13,]     0     0     0
## [14,]     0     0     0
## [15,]     0     0     0
## [16,]     0     0     0
```

```
## [17,]    0    0    0
## [18,]    0    0    0
## [19,]    0    0    0
## [20,]    0    0    0
## [21,]    0    0    0
## [22,]    0    0    0
## [23,]    0    0    0
## [24,]    0    0    0
## [25,]    0    0    0
## [26,]    0    0    0
## [27,]    0    0    0
## [28,]    0    0    0
```

```
#check for number 4
newmatrix4 <- accessMatrix(mnist$train$images[3,])
newmatrix4
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##  [2,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##  [3,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##  [4,]    0    0    0    0    0    0    0    0    0     0     0     0   150
##  [5,]    0    0    0    0    0    0    0    0    0     0     0   119   253
##  [6,]    0    0    0    0    0    0    0    0    0     0     0   177   237
##  [7,]    0    0    0    0    0    0    0    0    0     0     0   177   207
##  [8,]    0    0    0    0    0    0    0    0    0     0     0   177   207
##  [9,]    0    0    0    0    0    0    0    0    0     0     0   177   207
## [10,]    0    0    0    0    0    0    0    0    0     0     0   177   253
## [11,]    0    0    0    0    0    0    0    0    0     0     0    98   254
## [12,]    0    0    0    0    0    0    0    0    0     0     0    56   250
## [13,]    0    0    0    0    0    0    0    0    0     0     0     0   240
## [14,]    0    0    0    0    0    0    0    0    0     0     0     0   198
## [15,]    0    0    0    0    0    0    0    0    0     0     0     0   143
## [16,]    0    0    0    0    0    0    0    0    0     0     0     0    91
## [17,]    0    0    0    0    0    0    0    0    0     0     0     0    28
## [18,]    0    0    0   96  169  169  169  169  169   169   169   102     5
## [19,]    0    0    0  254  255  254  254  255  254   254   254   254   233
## [20,]    0    0    0  153  153  153   96   94   57    57   137   220   250
## [21,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [22,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [23,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [24,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [25,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [26,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [27,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [28,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##       [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
##  [1,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [2,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [3,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [4,]   159   159   159   120    46     0     0     0     0     0     0     0
##  [5,]   254   254   254   254   245   222   220   126    62     0     0     0
##  [6,]    85    67   120   163   163   163   163   163    81     0     0     0
##  [7,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [8,]     0     0     0     0     0     0     0     0     0     0     0     0
```

```
##  [9,]     0    0    0    0    0    0    0    0    0    0    0    0
## [10,]    47    0    0    0    0    0    0    0    0    0    0    0
## [11,]    49    0    0    0    0    0    0    0    0    0    0    0
## [12,]   116    0    0    0    0    0    0    0    0    0    0    0
## [13,]   144    0    0    0    0    0    0    0    0    0    0    0
## [14,]   150    0    0    0    0    0    0    0    0    0    0    0
## [15,]   241    0    0    0    0    0    0    0    0    0    0    0
## [16,]   243   14    0    0    0    0    0    0    0    0    0    0
## [17,]   234   86    0    0    0    0    0    0    0    0    0    0
## [18,]   179  178    0    0    0    0    0    0    0    0    0    0
## [19,]   241  248  163   23    0    0    0    0    0    0    0    0
## [20,]   252  254  254  231  198  183   27    2    0    0    0    0
## [21,]    40   91  216  254  254  254  254  153  120   67    0    0
## [22,]     0    0   16   29   56  125  162  210  180  232    0    0
## [23,]     0    0    0    0    0    0    0   40   39   39    0    0
## [24,]     0    0    0    0    0    0    0    0    0    0    0    0
## [25,]     0    0    0    0    0    0    0    0    0    0    0    0
## [26,]     0    0    0    0    0    0    0    0    0    0    0    0
## [27,]     0    0    0    0    0    0    0    0    0    0    0    0
## [28,]     0    0    0    0    0    0    0    0    0    0    0    0
##       [,26] [,27] [,28]
##  [1,]     0    0    0
##  [2,]     0    0    0
##  [3,]     0    0    0
##  [4,]     0    0    0
##  [5,]     0    0    0
##  [6,]     0    0    0
##  [7,]     0    0    0
##  [8,]     0    0    0
##  [9,]     0    0    0
## [10,]     0    0    0
## [11,]     0    0    0
## [12,]     0    0    0
## [13,]     0    0    0
## [14,]     0    0    0
## [15,]     0    0    0
## [16,]     0    0    0
## [17,]     0    0    0
## [18,]     0    0    0
## [19,]     0    0    0
## [20,]     0    0    0
## [21,]     0    0    0
## [22,]     0    0    0
## [23,]     0    0    0
## [24,]     0    0    0
## [25,]     0    0    0
## [26,]     0    0    0
## [27,]     0    0    0
## [28,]     0    0    0
```

```r
#check for number 5
newmatrix5 <- accessMatrix(mnist$train$images[1,])
newmatrix5
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
```

```
##  [1,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##  [2,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##  [3,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##  [4,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##  [5,]    0    0    0  136   55    0    0    0    0     0     0     0     0
##  [6,]    0    0    0  253  172    0    0    0    0     0     0     0     0
##  [7,]    0    0    0  253  226   18    0    0    0     0     0     0     0
##  [8,]    0    0    0  253  253  171    0    0    0     0     0     0     0
##  [9,]    0    0    0  212  253  219   23    0    0     0     0     0     0
## [10,]    0    0    0  135  253  253   66    0    0     0     0     0     0
## [11,]    0    0    0  132  253  253  213   24    0     0     0     0     0
## [12,]    0    0    0   16  244  253  253  114    0     0     0     0     0
## [13,]    0    0    0    0  133  253  253  221   39     0     0     0     0
## [14,]    0    0    0    0   11  195  253  253  148     0     0     0     0
## [15,]    0    0    0    0    0   80  253  253  229    46     0     0    45
## [16,]    0    0    0    0    0    9  198  253  253   130     0    16   186
## [17,]    0    0    0    0    0    0   81  253  253   183     0    93   253
## [18,]    0    0    0    0    0    0    2  201  253   253   249   252   253
## [19,]    0    0    0    0    0    0    0   78  250   253   253   253   150
## [20,]    0    0    0    0    0    0    0    0  182   207   249   187    27
## [21,]    0    0    0    0    0    0    0    0    0     2    64     0     0
## [22,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [23,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [24,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [25,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [26,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [27,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [28,]    0    0    0    0    0    0    0    0    0     0     0     0     0
##       [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
##  [1,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [2,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [3,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [4,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [5,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [6,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [7,]     0     0     0     0     0     0     0     0     0     0     0     0
##  [8,]     0     0     0     0     0     0    18    49     0     0     0     0
##  [9,]     0     0     0     0     0    80   219   238    30     0     0     0
## [10,]     0     0     0     0    14   156   253   253    36     0     0     0
## [11,]     0     0     0     0     1   107   253   253    94     0     0     0
## [12,]     0     0    11   139   154   253   253   253   154     0     0     0
## [13,]     0    35   190   253   253   253   253   253   170     3     0     0
## [14,]    81   241   253   190    90   205   253   253   253    18     0     0
## [15,]   240   225    70     2     0    11   198   253   253    18     0     0
## [16,]   253   160     0     0     0     0   182   253   253    18     0     0
## [17,]   253   108     0     0     0    43   247   253   253   126     0     0
## [18,]   119     1     0     0     0   154   241   251   253   136     0     0
## [19,]    25     0     0     0     0     0     0    93   225   175     0     0
## [20,]     0     0     0     0     0     0     0    82   172    26     0     0
## [21,]     0     0     0     0     0     0     0    82   253   166     0     0
## [22,]     0     0     0     0     0     0     0    56   242   255     0     0
## [23,]     0     0     0     0     0     0     0    39   195   247     0     0
## [24,]     0     0     0     0     0     0     0     0    64   127     0     0
## [25,]     0     0     0     0     0     0     0     0     0     0     0     0
```

```
## [26,]    0    0    0    0    0    0    0    0    0    0    0    0
## [27,]    0    0    0    0    0    0    0    0    0    0    0    0
## [28,]    0    0    0    0    0    0    0    0    0    0    0    0
##       [,26] [,27] [,28]
##  [1,]    0    0    0
##  [2,]    0    0    0
##  [3,]    0    0    0
##  [4,]    0    0    0
##  [5,]    0    0    0
##  [6,]    0    0    0
##  [7,]    0    0    0
##  [8,]    0    0    0
##  [9,]    0    0    0
## [10,]    0    0    0
## [11,]    0    0    0
## [12,]    0    0    0
## [13,]    0    0    0
## [14,]    0    0    0
## [15,]    0    0    0
## [16,]    0    0    0
## [17,]    0    0    0
## [18,]    0    0    0
## [19,]    0    0    0
## [20,]    0    0    0
## [21,]    0    0    0
## [22,]    0    0    0
## [23,]    0    0    0
## [24,]    0    0    0
## [25,]    0    0    0
## [26,]    0    0    0
## [27,]    0    0    0
## [28,]    0    0    0
```

Feature Definition

You are allowed to use only 2 features. Notice that you need to calculate those features directly from dataset. Make sure to describe what those features represent and why you chose them. Are those features capturing any intuition that you have about distinguishing those two digits?

For our focus features, we will look at symmetry over the top and bottom halves of the image, and the level of linearity that a 4 has vs a 3.

Symmetry

Starting with symmetry, we can see that a 3 is far more symmetrical between top and bottom halves than a 4 is. Thus, we will be looking at the number of pixels in the top half of the image divided by the number of pixels in the bottom half of the image, and the closer that value is to 1, the more symmetrical the image is, and the more likely the image is a 3.

```
#Calculating the symmetry of the upper quadrant

symmetry1 <- function(dat, newmatrix){
```

```
  sum <- 0
  for(x in 1:28){
    for(y in 1:14){
      sum = sum + newmatrix[x,y]
    }
  }
  sum
}
#Upper Quadrant symmetry for numbers 3(new matrix3) and number 4(newmatrix4)
symmetry1(mnist$train$images[8,], newmatrix3)
```

```
## [1] 18606
```

```
symmetry1(mnist$train$images[3,], newmatrix4)
```

```
## [1] 11587
```

```
#Calculating the symmetry of the lower quadrant
symmetry2 <- function(dat, newmatrix){
  sum <- 0
  for(x in 1:28){
    for(y in 15:28){
      sum = sum + newmatrix[x,y]
    }
  }
  sum
}
#Lower Quadrant symmetry for numbers 3(new matrix3) and number 4(newmatrix4)
symmetry2(mnist$train$images[8,], newmatrix3)
```

```
## [1] 17261
```

```
symmetry2(mnist$train$images[10,], newmatrix4)
```

```
## [1] 7856
```

Now that we have seen that it works for individual values, indices, and labels that represent 3 and 4, let's move on to see if it works generally. Here is the symmetry function we came up with:

```
ratio_calc <- function(index_of_minst){
  ratio <- c()
  y <- c()
  for (x in 1:500){
    matrix_group <- accessMatrix(mnist$train$images[index_of_minst[x],])
    upper_quadrant <- symmetry1(mnist$train$images[index_of_minst[x],], matrix_group)
    lower_quadrant <- symmetry2(mnist$train$images[index_of_minst[x],], matrix_group)
    ratio[x] = upper_quadrant/lower_quadrant
    y = 3
  }
  ratio
}

final_3 <- tibble(indeces = indeces$index_of3, ratio = ratio_calc(index_of3))
final_3 <- final_3%>%
  mutate(y = 3)
final_3
```

```
## # A tibble: 500 x 3
##    indeces ratio     y
##      <int> <dbl> <dbl>
##  1       8 1.08      3
##  2      11 0.882     3
##  3      13 1.22      3
##  4      28 0.967     3
##  5      31 1.02      3
##  6      45 1.07      3
##  7      50 0.822     3
##  8      51 0.856     3
##  9      75 0.951     3
## 10      87 0.784     3
## # ... with 490 more rows
```

```
final_4 <- tibble(indeces = indeces$index_of4, ratio = ratio_calc(index_of4))
final_4 <- final_4%>%
  mutate(y = 4)
final_4
```

```
## # A tibble: 500 x 3
##    indeces ratio     y
##      <int> <dbl> <dbl>
##  1       3  1.47     4
##  2      10  1.39     4
##  3      21  1.21     4
##  4      27  1.12     4
##  5      54  1.33     4
##  6      59  1.10     4
##  7      61  1.37     4
##  8      62  1.38     4
##  9      65  1.46     4
## 10      90  1.22     4
## # ... with 490 more rows
```

```
final_5 <- tibble(indeces = indeces$index_of5, ratio = ratio_calc(index_of5))
final_5 <- final_5%>%
  mutate(y = 5)
final_5
```

```
## # A tibble: 500 x 3
##    indeces ratio     y
##      <int> <dbl> <dbl>
##  1       1 1.09      5
##  2      12 1.10      5
##  3      36 0.986     5
##  4      48 1.32      5
##  5      66 0.897     5
##  6     101 1.03      5
##  7     133 1.11      5
##  8     139 1.38      5
##  9     146 1.19      5
## 10     174 1.06      5
## # ... with 490 more rows
```

```r
symmetry_final <- final_3%>%
  full_join(final_4)
symmetry_final
```

```
## # A tibble: 1,000 x 3
##    indeces ratio     y
##      <int> <dbl> <dbl>
##  1       8 1.08      3
##  2      11 0.882     3
##  3      13 1.22      3
##  4      28 0.967     3
##  5      31 1.02      3
##  6      45 1.07      3
##  7      50 0.822     3
##  8      51 0.856     3
##  9      75 0.951     3
## 10      87 0.784     3
## # ... with 990 more rows
```

Linearity

Next we will look at Linearity, because a 4 typically has a clear vertical line. A 3 should have less obvious of any vertical line, which should help with identifying, while looking at a different feature than before.

```r
Linear <- function(dat, newmatrix){
  min <- c()
  for(x in 1:28){
    sum <- 0
    for(y in 1:28){
      if(newmatrix[x,y] == 0){
        sum = sum + 1
      }
    }
    min[x] = sum
  }
  min(min)
}
Linear(mnist$train$images[8,], newmatrix3)
```

```
## [1] 10
```

```r
linear_final <- function(index_of_minst){
  minimum_values <- c()
  y <- c()
  for (x in 1:500){
    matrix_group <- accessMatrix(mnist$train$images[index_of_minst[x],])
    minimum_values[x] = Linear(mnist$train$images[index_of_minst[x,]], matrix_group)
  }
  print(minimum_values)
}

final_3_linear <- tibble(indeces = indeces$index_of3, linearity = linear_final(index_of3))
```

```
##   [1] 10  8 10  9 12 13 10 13  9 16 11 10 10 12 11 14  9 11 12 13 11 12 14  9 17
```

```
##  [26] 11  9 16 13 11 13 11 11 11 11 11 10 11  9 12 11 11 15 12 15 10 10 12 11 13
##  [51] 15 13 16 11 15 14 16 11 13 12 14 12 12 15 13  9 11 10 11 10 14  9 10 10 11
##  [76]  9 14  9 12  8 11  9 13 13 12 11 10 11 15 11 10 13 11 10 12  9 11 10 13  9
## [101] 14 14 15 10 12  9 14  9 13 11  9  8 10 10  8  8 10 10 17 11 12 13  8  9  8
## [126] 14 10 13 10  9 11 10 15 10 11 10 14 12  9 14 13 13 14 12 11 11  9 11 12 12
## [151] 11 13 12 10  9  9 14  9 10 14 11 11 12 11 15 11  8 13 15 13 13 12  9 12  9
## [176] 10 10  9 10 15 11 14 12 10 11 14 11  8 10 12 14 12  9  9 10 10 16 12  9 13
## [201] 16 10  9  9 11 13  9 12 11  9 12 10 10 12  9 14 13 12 12 11 12 11  9 12 12
## [226]  9 14 13 15 13 14 10 16 13 12 11 14 13 12 12 14 15 12 13 12 13 11 13 14 13
## [251]  9 12 11  8 11 15 11 15 10 12 11 11 10 13 13 14 15 13 13 14 17 11 12 14 13
## [276]  9 10 11 11  9 11 10 12 13 12 11 12 10 14 11 11  8 11 11 11 11 11  8 13 11
## [301] 11 11 12  9 13 13 13 11 13 11 11 10 11 11 10 12 10 14  9 11 10 12  9 13  9
## [326] 14 12 11 10  8 14 10 12 11 13 11  9 10 10 10 11 11 13  9 13 11 13 10 13 11
## [351] 12 11 12 12 12 12 12  8 14  9  9 12 13 10 12 16 12 13 11 11 13 14 10 13 16
## [376] 14 12 11  9 13 10  8 10 10  8  9 15 12 12  9 10  9 11 10  9  9  9  8 11  9
## [401]  9  8 11 11 11 11  8  9 16 10  9  8 14  8 13 10 12 11 14 10 12  8 11 12 11
## [426] 10 11 14 11 12 16 13 12 13  8  8 13 12 13 11 14 11 12  8 13  9 14 12 12 15
## [451] 14 11 12  9 14 12 10 11 14 11 10  9 11  9 11  8 15 11  8 16 10 12 11 10 11
## [476] 10 15 14  8 10 12 13 11 13 11 14 12 11 11  9  8 10 10 11  9 12 11 11 12  9
```

```r
final_3_linear <- final_3_linear%>%
  mutate(y = 3)
final_3_linear
```

```
## # A tibble: 500 x 3
##    indeces linearity     y
##      <int>     <dbl> <dbl>
##  1       8        10     3
##  2      11         8     3
##  3      13        10     3
##  4      28         9     3
##  5      31        12     3
##  6      45        13     3
##  7      50        10     3
##  8      51        13     3
##  9      75         9     3
## 10      87        16     3
## # ... with 490 more rows
```

```r
final_4_linear <- tibble(indeces = indeces$index_of4, linearity = linear_final(index_of4))
```

```
##   [1] 10 12  8 15 14  8 10 14 16 13  8 15 14 12 13 17 10 13  8 15  9 10 10 11 17
##  [26] 14 15 13 17 10  9 15 15 17 14 17 12 13 17 16 12 17 14 11 17 16 17  8  8 15
##  [51] 17 16 16 14 14 12 10 16 11 16 11  9 17 14 18 14 14 11 12  9 11 12  8 12 13
##  [76] 11 17  8 11 15 10  9 10 12 10 15  8 14 14 11 14  9  8  9 19  8  8  8 14 14
## [101] 14 14 14  9 12  8  8 14  9 13 13 13 16  8 11 14 10 15 10 16 16 14 14  9 14
## [126] 11 14 15  8 13  8  9 12 10 11  9  8 11  8  8  8  9 13  9 14 12 14 11 14 14
## [151] 18 10  8 13 10 15 15 15 14 12  9 15 13 17 15 10  8 16 11  9  8  8 14  8 14
## [176] 14  8 16 16 17  8 15  8 16  8 14 12 13 15 16 11 14 16 11  8  8 16 18  8  8
## [201] 13 11  8 15 14 10  8  8 12 16  9 16  8 10 12  8 14  9  9 11  8  9  8  9 12
## [226] 13 12 15 16 15 16 11  8 14  9  8 10 10 13 14 15 11 17 12 11  8 12  9  8  9
## [251] 10 13 15 12 13 13  8  8 11 14 10  8 17  9 13 14  8 10 10 10 13 13 10 13  8
## [276] 10 11 11 11 10 14  8  8  8 12 12  8 14  8 18 10 11 14 14 13 17  8  8  9 12
## [301]  8 12 19 15 15  8 16 15 15 11  9 13 11 10 16 14  9 13 10 12 14  8 13 12  9
## [326] 12  8 12 15 12 14 19 12 13 12  8 11 12 13 10 12 10 12 15  8 13 11 16 12  8
```

```
## [351] 14 10  9  8 11 13 14  8 10  8 10  8  8 13  8 14 15  9 12  9  9  9 14 14 14
## [376] 14 15 12 11 12  8 14 13 15  9 16 16  9 14  9 13 10 14 14 12 18 17  8 12 12
## [401]  8 10 13  8 15  8 18 13 12 12 14 17 16 11 11 14 15 16  8 17 14 14 11 13  8
## [426] 12 13 11 13 11 12 10 16 15 14 17 14 13 13  8 14  8 13 12 13 12 13 10 13 12
## [451] 14 12 16 11  8  8  8 15  9 14 11 14 14  8  8 15 12 15  9 13 15  8 15 12 15
## [476] 17  8 15 13 11  8  8 14  9  8 11 14  9 11 15  8 14 15 10  8 16 12 16 14 15
```

```
final_4_linear <- final_4_linear%>%
  mutate(y = 4)
final_4_linear
```

```
## # A tibble: 500 x 3
##    indeces linearity     y
##      <int>     <dbl> <dbl>
##  1       3        10     4
##  2      10        12     4
##  3      21         8     4
##  4      27        15     4
##  5      54        14     4
##  6      59         8     4
##  7      61        10     4
##  8      62        14     4
##  9      65        16     4
## 10      90        13     4
## # ... with 490 more rows
```

```
linear_final_tbl <- final_3_linear%>%
  full_join(final_4_linear)
linear_final_tbl
```

```
## # A tibble: 1,000 x 3
##    indeces linearity     y
##      <int>     <dbl> <dbl>
##  1       8        10     3
##  2      11         8     3
##  3      13        10     3
##  4      28         9     3
##  5      31        12     3
##  6      45        13     3
##  7      50        10     3
##  8      51        13     3
##  9      75         9     3
## 10      87        16     3
## # ... with 990 more rows
```

Will all this, now we put all the data into a combined table.

```
mnist_34 <- linear_final_tbl %>%
  bind_cols(symmetry_final)
mnist_34
```

```
## # A tibble: 1,000 x 6
##    indeces...1 linearity y...3 indeces...4 ratio y...6
##          <int>     <dbl> <dbl>       <int> <dbl> <dbl>
##  1           8        10     3           8 1.08      3
##  2          11         8     3          11 0.882     3
##  3          13        10     3          13 1.22      3
```

```
## 4         28         9     3          28 0.967     3
## 5         31        12     3          31 1.02      3
## 6         45        13     3          45 1.07      3
## 7         50        10     3          50 0.822     3
## 8         51        13     3          51 0.856     3
## 9         75         9     3          75 0.951     3
## 10        87        16     3          87 0.784     3
## # ... with 990 more rows
```

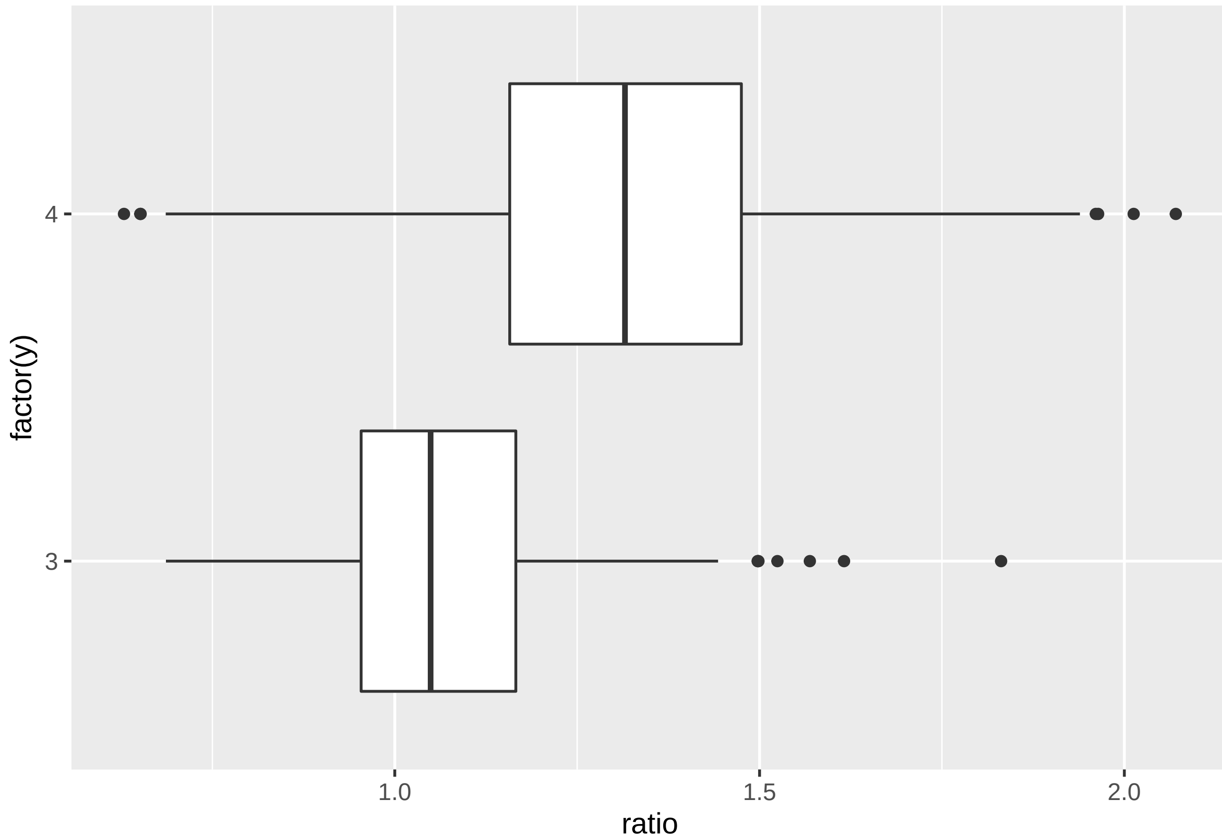
```
final.split <- initial_split(mnist_34, prop=0.8)
train.mnist_34 <- training(final.split)%>%
  mutate(y = as.factor(y...6))%>%
  mutate(x_1 = ratio)%>%
  mutate(x_2 = linearity)%>%
  select(x_1, x_2, y)
train.mnist_34
```

```
## # A tibble: 800 x 3
##      x_1   x_2 y
##    <dbl> <dbl> <fct>
##  1 1.07     11 3
##  2 0.909    11 3
##  3 1.17     14 3
##  4 0.975    12 3
##  5 1.38     14 4
##  6 1.04     11 3
##  7 0.965     9 4
##  8 1.07     13 3
##  9 1.11     14 4
## 10 0.788    11 4
## # ... with 790 more rows
```

```
test.mnist_34 <- testing(final.split)%>%
  mutate(y = as.factor(y...6))%>%
  mutate(x_1 = ratio)%>%
  mutate(x_2 = linearity)%>%
  select(x_1, x_2, y)
```

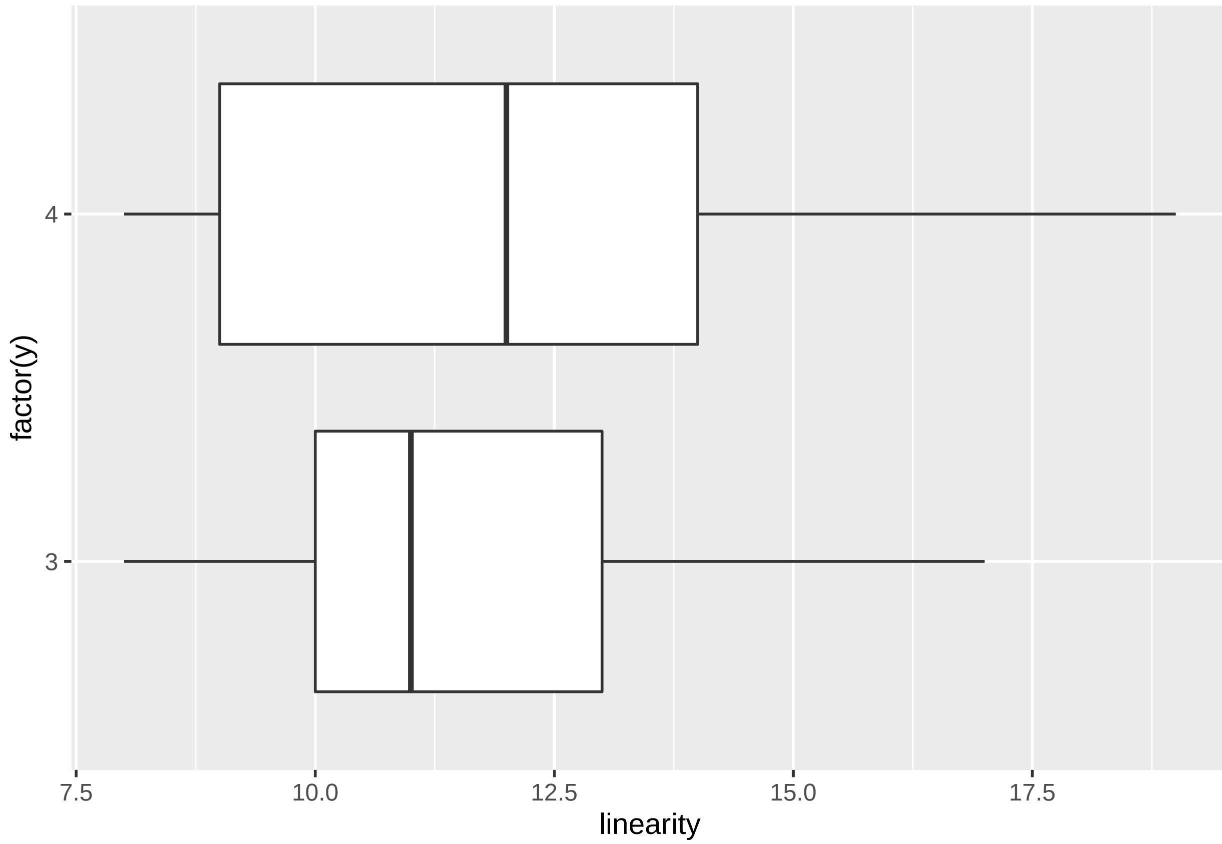
Here are some graphs that can help us better understand our distribution.

```
ggplot(symmetry_final, aes(x=ratio, y = factor(y)))+
  geom_boxplot()
```



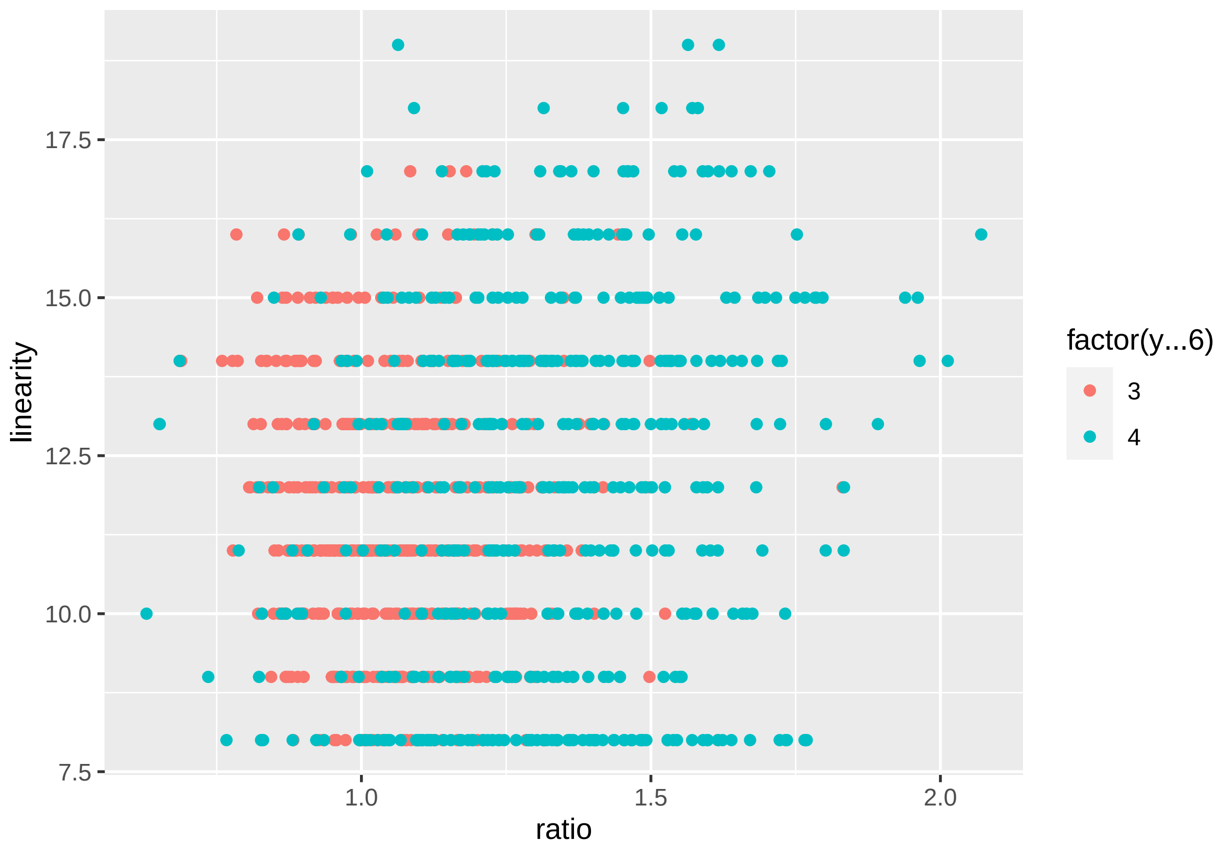
This box plot shows us how accurate the symmetry feature was, looking at the ratio of pixels from the top half to the bottom half. We can see that there's very little overlap between the numbers, and when identifying, a 3 will stick around a 1:1 ratio, showing symmetry, while the ratio for a 4 is either decently above or below 1.0 - typically above, as there should be more pixels in the top half than the bottom half, but sometimes below.

```
ggplot(linear_final_tbl, aes(x=linearity, y = factor(y)))+
  geom_boxplot()
```



Now, when looking at the linearity box plot, we can see that this is far less accurate of a feature for us to be looking at, as there's a lot of overlap between the numbers. The values along the x axis show us the minimum number of zeros in a row in the image, trying to find the most linear line. The 4s actually shows more linearity than a 3 generally, at least in these images, we believe because a lot of the images in this dataset have the vertical line in a 4 somewhat diagonal.

```
ggplot(mnist_34, aes(x=ratio, y = linearity, color = factor(y...6)))+
  geom_point()
```



This scatter plot is going more in depth with both features, seeing how well symmetry (on the x axis) and linearity (on the y axis) work together to identify the numbers. There's a significant amount of overlap, but the we can see that the symmetry feature really helps identify a 3 from a 4, and the linearity helps somewhat, but mainly with outliars.

Model Creation, Optimization and Selection

a) Create at least two different models for this classification and make sure to optimize the parameters those models have.

KNN Model

```
library(tidymodels)
library(kknn)
## devtools::install_github("KlausVigo/kknn")
tidymodels_prefer()

build_knn <- function (train.table, kVal) {
  knn.model <- nearest_neighbor(neighbors = kVal) %>%
    set_engine("kknn") %>%
    set_mode("classification")

  recipe <- recipe(y ~ x_1 + x_2, data=train.table)
```

```r
  knn.wflow <- workflow() %>%
    add_recipe(recipe) %>%
    add_model(knn.model)

  knn.fit <- fit(knn.wflow, train.table)
}

knn.model <- build_knn(train.mnist_34, 5)
```

Cross Validation

```r
knn.model.cv <- nearest_neighbor(neighbors = tune()) %>%
    set_engine("kknn") %>%
    set_mode("classification")

recipe <- recipe(y ~ x_1 + x_2, data=train.mnist_34)

knn.wf <- workflow() %>%
    add_recipe(recipe) %>%
    add_model(knn.model.cv)
knn.wf
```

```
## == Workflow ===================================================================
## Preprocessor: Recipe
## Model: nearest_neighbor()
##
## -- Preprocessor -------------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model ---------------------------------------------------------------------------
## K-Nearest Neighbor Model Specification (classification)
##
## Main Arguments:
##   neighbors = tune()
##
## Computational engine: kknn
```

b) Calculate the missclassification rates for both models and select the model with the lowest error rate.

```r
knn.final.fit <- predict(knn.model, test.mnist_34, type="prob")

pred34.test.tbl <- knn.model %>%
  augment(new_data = test.mnist_34)
pred34.test.tbl
```

```
## # A tibble: 200 x 6
##      x_1   x_2 y     .pred_class .pred_3 .pred_4
##    <dbl> <dbl> <fct> <fct>         <dbl>   <dbl>
##  1 1.22     10 3     4              0.12    0.88
##  2 0.967     9 3     3              0.64    0.36
##  3 1.02     10 3     3              1       0
##  4 0.989    12 3     3              1       0
##  5 0.977    13 3     3              1       0
```

```
##   6 0.930     12 3     3               0.72    0.28
##   7 1.30      16 3     4               0.2     0.8
##   8 1.04      11 3     3               0.72    0.28
##   9 1.09      10 3     3               1       0
##  10 1.32      11 3     4               0.48    0.52
## # ... with 190 more rows
```

```
accuracy(pred34.test.tbl, y, .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.695
```

```
conf_mat(pred34.test.tbl, truth = y, estimate = .pred_class)
```

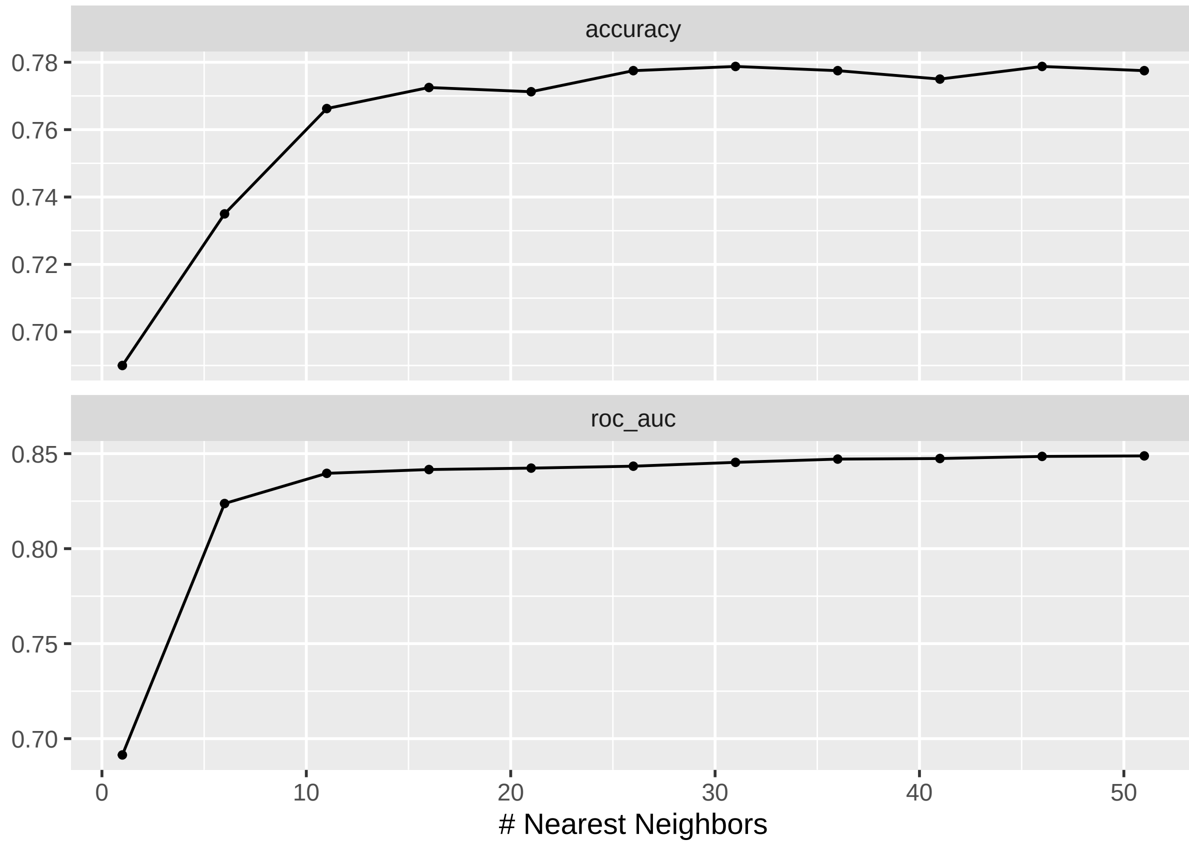
```
##           Truth
## Prediction  3  4
##          3 63 38
##          4 23 76
```

Looking at our results from the KNN Model, the missclassification rate is 0.75, and in the confusion matrix we can see the results played out in more detail. These were really cool results to get, as it shows that this model is pretty accurate for identifying the numbers. Now we'll look at the rates with the cross validation, and see if it helps at all.

```
set.seed(12345)
digits.folds <- vfold_cv(train.mnist_34, v = 10)

neighbours <- seq(1, 51, by = 5)
neighbors.tbl <- tibble(neighbours)
neighbors2.tbl <- grid_regular(neighbors(range = c(1, 51)), levels = 11)

tune.results <- tune_grid(
  object = knn.wf,
  resamples = digits.folds,
  grid = neighbors2.tbl
)
autoplot(tune.results)
```




```r
show_best(tune.results, metric = "accuracy")
```

```
## # A tibble: 5 x 7
##   neighbors .metric  .estimator  mean     n std_err .config
##       <int> <chr>    <chr>      <dbl> <int>   <dbl> <fct>
## 1        31 accuracy binary     0.779    10  0.0124 Preprocessor1_Model07
## 2        46 accuracy binary     0.779    10  0.0126 Preprocessor1_Model10
## 3        26 accuracy binary     0.778    10  0.0120 Preprocessor1_Model06
## 4        36 accuracy binary     0.778    10  0.0126 Preprocessor1_Model08
## 5        51 accuracy binary     0.778    10  0.0115 Preprocessor1_Model11
```

```r
best.neighbor <- select_best(tune.results, metric = "accuracy")
knn.final.wf <- finalize_workflow(knn.wf, best.neighbor)
knn.final.fit_cv <- fit(knn.final.wf, train.mnist_34)

predict(knn.final.fit_cv, test.mnist_34, type="prob")
```

```
## # A tibble: 200 x 2
##    .pred_3 .pred_4
##      <dbl>   <dbl>
##  1  0.580   0.420
##  2  0.854   0.146
##  3  0.954   0.0458
##  4  0.895   0.105
##  5  0.840   0.160
##  6  0.876   0.124
##  7  0.0926  0.907
```

```
## 8  0.849   0.151
## 9  0.795   0.205
## 10 0.575   0.425
## # ... with 190 more rows
```

```
pred34_cv.test.tbl <- knn.final.fit_cv %>%
  augment(new_data = test.mnist_34)
pred34_cv.test.tbl
```

```
## # A tibble: 200 x 6
##      x_1   x_2 y     .pred_class .pred_3 .pred_4
##    <dbl> <dbl> <fct> <fct>         <dbl>   <dbl>
##  1 1.22     10 3     3            0.580   0.420
##  2 0.967     9 3     3            0.854   0.146
##  3 1.02     10 3     3            0.954   0.0458
##  4 0.989    12 3     3            0.895   0.105
##  5 0.977    13 3     3            0.840   0.160
##  6 0.930    12 3     3            0.876   0.124
##  7 1.30     16 3     4            0.0926  0.907
##  8 1.04     11 3     3            0.849   0.151
##  9 1.09     10 3     3            0.795   0.205
## 10 1.32     11 3     3            0.575   0.425
## # ... with 190 more rows
```

```
accuracy(pred34_cv.test.tbl, y, .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary          0.74
```

```
conf_mat(pred34_cv.test.tbl, truth = y, estimate = .pred_class)
```

```
##           Truth
## Prediction  3  4
##          3 72 38
##          4 14 76
```

Now with Cross Validation, the missclassification rate is 0.785, and the confusion matrix. This shows us that the Cross Validation model is more accurate than the KNN Model! It is only slightly better, but definitely worth it - this is our superior model, and now we'll plot these probabilities.

Visualization

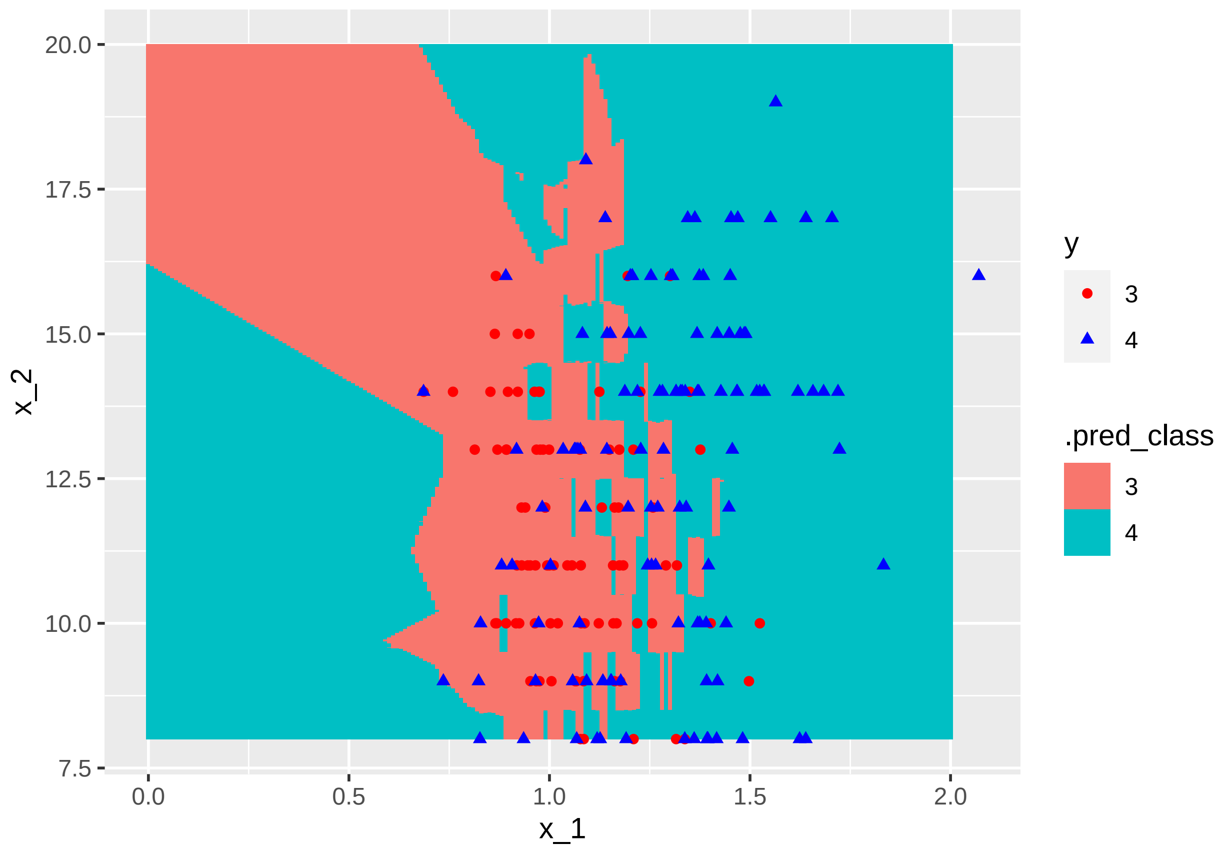
Plot the probabilities across a grid and the decision boundary for your selected model.

We selected the Cross Validation model, as it had better results, and here is our plot.

```
plot_boundary <- function(fit, test.tbl, delta){
  grid.tbl <- expand_grid(x_1=seq(0,2, by=delta),
                          x_2=seq(8,20, by=delta))

  augment(fit, grid.tbl)%>%
    ggplot() +
      geom_raster(aes(x_1, x_2, fill = .pred_class)) +
      geom_point(data=test.tbl, aes(x=x_1, y=x_2, color=y, shape=y))+
```

```r
      scale_color_manual(values=c("red","blue"))
}
plot_boundary(knn.model, test.mnist_34, 0.01)
```



Changing things up

a) Create a new dataset that includes your two chosen digits and the digit 5. Create training and testing datasets that include 5 and your two given digits.

b) Calculate the same 2 features for this new testing and training dataset.

```r
final_5_linear <- tibble(indeces = indeces$index_of5, linearity = linear_final(index_of5))
```

```
##   [1] 13 18 16 18 16 15  8 13 17 17 15 13 17 14 11 14 10 14 14 14 12 14 16 16 18
##  [26] 11 16 16 14 16 15 15 17 14 16 17 16 10 12 14 12 14 17 15 15 17 14 16 14 10
##  [51] 18 13 16 12 12 12 14 10 13 13 12 13 12 13 12 12 13 16 12 15 18 12 11 11 13
##  [76] 14 13 11 18 14 17 15 15  9 17 15 17 13 14 13 13 17 16 10 15 13 14 14 16 12
## [101] 13 15 17 16 11 12 16 12  8  8 17 11 14 12 16 13 12 11 11 16 15 15  8 11 14
## [126] 13 16 12 14 19 12 17 12 17 13 17 12 14 13 11 13 14 13 17 17 17 12 13 17 18
## [151] 10 16 15 14 11 16 12 14 14 16 17 14 16 14 13 11 12 12 12 15 12 11 11 12 17
## [176] 14 13 10 15 11 13 15 10 14 13 16 11 13 13 14 13 12 14 11 14 16 12 10 13  9
## [201] 12 10 13  9 12 13 12 16 13 16 14 13 15 14 14 16 15 11 12 14 17 12 14 14 13
## [226] 17 12 15 14 15 13 18 17  8 10 12  9 13 16 13 18 12 15 16 14 17 13 19 15 13
## [251] 13 13 14 14 16 15 13 16 16 13 16 13 10 13 15 14 16 15 16 18 10 11 14 16 17
## [276] 17 14 14 18 13 15 15 15 16 14 13 14 12 13 10 17 11 14 11 12 11 10 12 15 14
```

```
## [301] 13 15 13 14  9 12 12 15 13 11 13 16 16 17 17 13 17 16 16 14 17 14 13 15 12
## [326] 13 16 15 12 15 16 13 11 13 15 13 11 14 13 13 17 11 15 15 16 15 17 11 13 13
## [351] 11 14 14 15 13 12 15 17 13 15 16 17 14 14 16 16 13 12 13 16 11 14 11 14 10
## [376] 14 12 14 14 14 13 16 16 14 17 15 14 13 16 11 12 11 13 12 14 13 16 13 18 16
## [401] 10 11 13 14 11 12 13 11 11 11 15 13 15 12 14 12 16 16 12 11 11 14 11 11 12
## [426]  8 14 14 15 15 16 13 15 13 13 11 12 12 17 13 13 15 14 19 14 15 17 15 13 14
## [451] 12 14 18 12 18 18 13 14 14 11 13 18 14 11 14 16 12 13 14 13 11 11 15 14 13
## [476] 15 14  9 17 15 13 16 16 14 14 14 11 11 18 16 12 14 11 16 13 14 16 20 13 15
```

```
final_5_linear <- final_5_linear%>%
  mutate(y = 5)
final_5_linear
```

```
## # A tibble: 500 x 3
##    indeces linearity     y
##      <int>     <dbl> <dbl>
##  1       1        13     5
##  2      12        18     5
##  3      36        16     5
##  4      48        18     5
##  5      66        16     5
##  6     101        15     5
##  7     133         8     5
##  8     139        13     5
##  9     146        17     5
## 10     174        17     5
## # ... with 490 more rows
```

```
linear_final_tbl_3_4_5 <- linear_final_tbl%>%
  full_join(final_5_linear)
linear_final_tbl_3_4_5
```

```
## # A tibble: 1,500 x 3
##    indeces linearity     y
##      <int>     <dbl> <dbl>
##  1       8        10     3
##  2      11         8     3
##  3      13        10     3
##  4      28         9     3
##  5      31        12     3
##  6      45        13     3
##  7      50        10     3
##  8      51        13     3
##  9      75         9     3
## 10      87        16     3
## # ... with 1,490 more rows
```

```
symmetry_final_3_4_5 <- symmetry_final%>%
  full_join(final_5)
symmetry_final_3_4_5
```

```
## # A tibble: 1,500 x 3
##    indeces ratio     y
##      <int> <dbl> <dbl>
##  1       8 1.08      3
##  2      11 0.882     3
```

```
## 3       13 1.22      3
## 4       28 0.967     3
## 5       31 1.02      3
## 6       45 1.07      3
## 7       50 0.822     3
## 8       51 0.856     3
## 9       75 0.951     3
## 10      87 0.784     3
## # ... with 1,490 more rows
```

```
mnist_345 <- linear_final_tbl_3_4_5%>%
  bind_cols(symmetry_final_3_4_5)
mnist_345
```

```
## # A tibble: 1,500 x 6
##    indeces...1 linearity y...3 indeces...4 ratio y...6
##          <int>     <dbl> <dbl>       <int> <dbl> <dbl>
##  1           8        10     3           8 1.08      3
##  2          11         8     3          11 0.882     3
##  3          13        10     3          13 1.22      3
##  4          28         9     3          28 0.967     3
##  5          31        12     3          31 1.02      3
##  6          45        13     3          45 1.07      3
##  7          50        10     3          50 0.822     3
##  8          51        13     3          51 0.856     3
##  9          75         9     3          75 0.951     3
## 10          87        16     3          87 0.784     3
## # ... with 1,490 more rows
```

```
final.split_345 <- initial_split(mnist_345, prop=0.8)
train.mnist_345 <- training(final.split_345)%>%
  mutate(y = as.factor(y...6))%>%
  mutate(x_1 = ratio)%>%
  mutate(x_2 = linearity)%>%
  select(x_1, x_2, y)
train.mnist_345
```

```
## # A tibble: 1,200 x 3
##      x_1   x_2 y
##    <dbl> <dbl> <fct>
##  1 1.02     12 3
##  2 1.32     11 3
##  3 0.804    14 5
##  4 0.965     9 4
##  5 0.969    16 5
##  6 0.864    15 3
##  7 1.20     15 4
##  8 1.12     10 5
##  9 1.08     11 3
## 10 0.966    11 3
## # ... with 1,190 more rows
```

```
test.mnist_345 <- testing(final.split_345)%>%
  mutate(y = as.factor(y...6))%>%
  mutate(x_1 = ratio)%>%
  mutate(x_2 = linearity)%>%
```

```
  select(x_1, x_2, y)
```

c) Calculate the missclassification rate on this new dataset. Create also the confusion matrix and comment on what digits seem to get confused more and why.

```
build_knn_345 <- function (train.table, kVal) {
  knn.model <- nearest_neighbor(neighbors = kVal) %>%
    set_engine("kknn") %>%
    set_mode("classification")

  recipe <- recipe(y ~ x_1 + x_2, data=train.table)

  knn.wflow <- workflow() %>%
    add_recipe(recipe) %>%
    add_model(knn.model)

  knn.fit <- fit(knn.wflow, train.table)
}

knn.model_345 <- build_knn(train.mnist_345, 5)

knn.final.fit_345 <- predict(knn.model_345, test.mnist_345, type="prob")

pred345.test.tbl <- knn.model_345 %>%
  augment(new_data = test.mnist_345)
pred345.test.tbl
```

```
## # A tibble: 300 x 7
##      x_1   x_2 y     .pred_class .pred_3 .pred_4 .pred_5
##    <dbl> <dbl> <fct> <fct>         <dbl>   <dbl>   <dbl>
##  1 0.945    11 3     3              0.72    0       0.28
##  2 0.996    11 3     3              0.72    0.28    0
##  3 0.958     9 3     3              0.76    0.24    0
##  4 1.30     16 3     5              0       0.04    0.96
##  5 1.05     13 3     5              0.12    0       0.88
##  6 1.11     11 3     3              0.52    0.36    0.12
##  7 0.844     9 3     3              1       0       0
##  8 0.995    15 3     5              0.24    0       0.76
##  9 1.17     10 3     3              0.68    0.32    0
## 10 0.836    14 3     3              0.6     0       0.4
## # ... with 290 more rows
```

```
accuracy(pred345.test.tbl, y, .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.517
```

```
conf_mat(pred345.test.tbl, truth = y, estimate = .pred_class)
```

```
##           Truth
## Prediction  3  4  5
##          3 53 30 27
##          4 18 46 14
##          5 33 23 56
```

Once we add in the 5, our accuracy with the KNN Model drops significantly with a missclassification rate of 0.517.As we can see from the confusion matrix, the 3s and 5s are most often confused, which makes sense because with the way we calculated symmetry, a 5 would also be seen as quite symmetrical. 4s were the least confused, but still easily confused.As we can see, the numbers are still more accurate than not, but only slightly. Now we'll see the plot that should help us visualize.

d) Plot the probabilities across a grid and the decision boundary for your model.

```
plot_boundary3 <- function(fit, test.tbl, delta){
  grid.tbl <- expand_grid(x_1=seq(0,2, by=delta),
                          x_2=seq(8,20, by=delta))

  augment(fit, grid.tbl)%>%
    ggplot() +
      geom_raster(aes(x_1, x_2, fill = .pred_class)) +
      geom_point(data=test.tbl, aes(x=x_1, y=x_2, color=y, shape=y))+
      scale_color_manual(values=c("red","green","blue"))
}
plot_boundary3(knn.model_345, test.mnist_345, 0.01)
```