Challenge #1

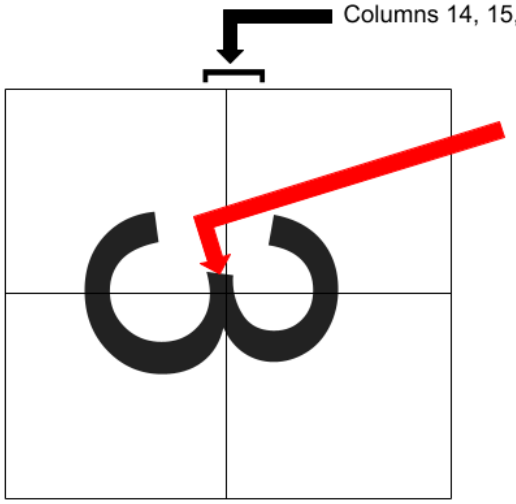
Anike, Jasmine & Katelyn

10/13/23

Feature Definition

In the introduction of machine learning and decision making algorithms, the digit classifier has been introduced to give students a good starting point on how to create an algorithm and conduct analysis to classify between two handwritten digits. In class, we looked at the number of dark pixels in the upper left and bottom right quadrants to classify between a 2 and a 7. Similarly, to distinguish between a 3 and a 7, one of our features is the average left border of the middle rows (14-16) of a digit image (or columns 14-16 in the corresponding matrix). Our second feature will be the summed intensity of the pixels in the bottom right quadrant of the image, or the sum of all of the entries from rows 15-28 and columns 1-14 in the matrix of that image. Below are the graphical depictions and more detailed explanations of what each feature looks like, as well as the code for how they are calculated, where the input for each function is a vector that denotes a specific image:

FEATURE 1 - LEFT CENTER BORDER



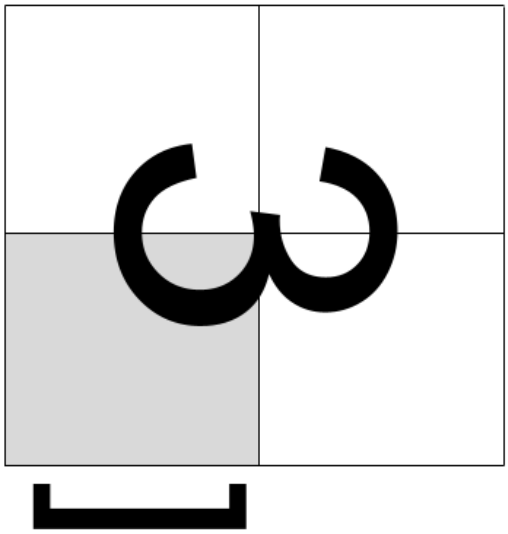
This is what we are considering the center left border. On this digit, since it occurs slightly before the middle (rows 14 and 15), it might have a center left border value of about 11.

The reason that we chose this as a feature is because in theory, a seven will have a center left border that is a higher value than a three will, given the straightness of a 7 opposed to the curves and angles of a 3.

Left center border is calculated by going through columns 14, 15, and 16 of a matrix and storing the row number of the first non-zero cell. The average of the three row numbers is calculated and returned from the function.

This is how the image will be oriented when the border is calculated, which is why we go through the middle three columns instead of rows.

FEATURE 2 - TOTAL LOWER RIGHT QUADRANT DARKNESS



This is the quadrant (in gray) that our function looks at. The image is oriented this way because this is the how the function will be "seeing" the image in the matrix. As a result, our function goes through the second half of rows (rows 15-28) and the first half of columns (columns 1-14).

The function goes through each row and column in this quadrant and sums the pixel value for each cell (from 0-255). In the end, the sum of the quadrant's pixel values is returned. Since the end result is a sum, the returned value will likely be quite a large number.

The reason we chose this feature is because 3s, in theory, should cover more space in this quadrant than 7s do, given the straightness of a 7 compared to the roundness of a 3.

```
# FEATURE #1 FUNCTION

get_center_left_border <- function(image_vector){
  image_mat <- matrix(image_vector, nrow=28)
  image_mat_1 <- image_mat[,28:1]
  border_14 <- -1
  border_15 <- -1
  border_16 <- -1

  for(i in 1:28){ #go through rows in 14-16 col (because of rotation of image when plotted)
    if(image_mat_1[i,14] != 0 & border_14 == -1){ #if first nonzero col, set to border
      border_14 = i
    }
    if(image_mat_1[i,15] != 0 & border_15 == -1){
      border_15 = i
    }
    if(image_mat_1[i,16] != 0 & border_16 == -1){
      border_16 = i
    }

  }

  avg_border <- (border_14 + border_15 + border_16) / 3
}
```

```r
# FEATURE #2 FUNCTION

bottom_right_quadrant <- function(image) {
  a_digit <- matrix(image, nrow = 28) [,28:1]
  br_quadrant <- a_digit[15:28, 1:14]
  return (sum(br_quadrant))
}
```

We believe that the detailed graphical depictions and explanation of the two features we provided are sufficient to build a highly effective 3-and-7 classifier. The process takes in a good amount of distinct characteristics of the two digits and different scenarios when they are handwritten.

Dataset Creation

In this section of code, we are building the training and the testing data sets. The way that both the testing and the training sets are created is almost exactly the same; the only differences are the amount of images randomly selected (800 for training and 200 for testing) and the set seed used for each random selection. Broadly speaking, the way that the data sets were created was by pulling all of the 3s and 7s images from the respective mnist data set (training or testing) and combining them into a new matrix, randomly selecting the necessary number of rows (800 or 200), calculating both of the features for every row that was selected, and then storing the information about each row/digit into a table (whether the digit is a 3 or a 7, its value of Feature #1, its value of Feature #2, and its row in the matrix, for plotting purposes). These tables are the training and testing data sets. The image vectors for the digits that were randomly selected are maintained in a matrix so that any digit can be plotted later on. The code to build the training and testing data sets is as follows:

```r
# BUILDING THE TRAINING DATA SET

# filtering the mnist training data set to create a matrix where all of the
# rows correspond to the digit 3
threes_train <- get_image_train_digit(3)

# checking the dimension of the digit 3 matrix; there are a total of 6,131 3s
# in the mnist training data set (denoted by the number of rows in the matrix)
dim(threes_train)
```

```
## [1] 6131  784
```

```r
# filtering the mnist training data set to create a matrix where all of the
# rows correspond to the digit 7
sevens_train <- get_image_train_digit(7)

# checking the dimension of the digit 7 matrix; there are a total of 6,265 7s
# in the mnist training data set (denoted by the number of rows in the matrix)
dim(sevens_train) # there are 6,265 7s in the matrix (number of rows)
```

```
## [1] 6265  784
```

```r
# creating a vector that denotes what digit a row corresponds to; this will be
# helpful for when the digit 3 matrix and digit 7 matrix are combined
digit_train <- rep(c(3, 7), times = c(6131, 6265))

# setting the seed so that the randomly selected values can be duplicated later
set.seed(22)

# first line: combining the digit 3 and digit 7 matrices and making them a tibble
# second line: adding the vector that denotes what digit a row is to the tibble
# third line: selecting 800 rows from the tibble at random
training_37 <- as_tibble(rbind(threes_train, sevens_train)) %>%
  mutate(y = digit_train) %>%
  sample_n(800)

# selecting the column vector from the tibble that denotes what digit a row
# corresponds to and storing it; this will be a column in the training data set
digit_37_train <- training_37$y

# turning the tibble that was created into a matrix and storing it for later;
# this will be useful for calculating the features as well as plotting specific
# digits as images later on
training_37_matrix <- as.matrix(training_37 %>% select(1:784))

# verifying that the matrix has 800 rows and 784 columns
dim(training_37_matrix)
```

```
## [1] 800 784
```

```r
# setting up an empty vector to later store the values of Feature #1 in
train_x1 <- vector(mode = "double", length = 800)

# running a for loop with our Feature #1 function to fill in the empty vector
# with the values of Feature #1 for the 800 observations in the training data set
for (i in 1:800) {
  train_x1[i] <- get_center_left_border(training_37_matrix[i,]) # filling the vector
}

# setting up an empty vector to later store the values of Feature #2 in
train_x2 <- vector(mode = "double", length = 800) # empty vector to store x_2 in

# running a for loop with our Feature #2 function to fill in the empty vector
# with the values of Feature #2 for the 800 observations in the training data set
for (i in 1:800) {
  train_x2[i] <- bottom_right_quadrant(training_37_matrix[i,]) # filling the vector
}

# creating the official training data set; there is a column for the digit (3 or 7),
# the value of Feature #1 (x_1), the value of Feature #2 (x_2), and the row
# number that a given observation corresponds to in the matrix that was stored
train_37_tbl <- tibble(y = as.factor(digit_37_train), x_1 = train_x1,
                       x_2 = train_x2, mat_row_num = 1:800)
```

```r
# BUILDING THE TESTING DATA SET

# filtering the mnist testing data set to create a matrix where all of the
# rows correspond to the digit 3
threes_test <- get_image_test_digit(3)

# checking the dimension of the digit 3 matrix; there are a total of 1,010 3s
# in the mnist testing data set (denoted by the number of rows in the matrix)
dim(threes_test)
```

```
## [1] 1010  784
```

```r
# filtering the mnist testing data set to create a matrix where all of the
# rows correspond to the digit 7
sevens_test <- get_image_test_digit(7)

# checking the dimension of the digit 7 matrix; there are a total of 1,028 7s
# in the mnist testing data set (denoted by the number of rows in the matrix)
dim(sevens_test)
```

```
## [1] 1028  784
```

```r
# creating a vector that denotes what digit a row corresponds to; this will be
# helpful for when the digit 3 matrix and digit 7 matrix are combined
digit_test <- rep(c(3, 7), times = c(1010, 1028))

# setting the seed so that the randomly selected values can be duplicated later
set.seed(7)

# first line: combining the digit 3 and digit 7 matrices and making them a tibble
# second line: adding the vector that denotes what digit a row is to the tibble
# third line: selecting 200 rows from the tibble at random
testing_37 <- as_tibble(rbind(threes_test, sevens_test)) %>%
  mutate(y = digit_test) %>%
  sample_n(200)

# selecting the column vector from the tibble that denotes what digit a row
# corresponds to and storing it; this will be a column in the testing data set
digit_37_test <- testing_37$y

# turning the tibble that was created into a matrix and storing it for later;
# this will be useful for calculating the features as well as plotting specific
# digits as images later on
testing_37_matrix <- as.matrix(testing_37 %>% select(1:784))

# verifying that the matrix has 200 rows and 784 columns
dim(testing_37_matrix)
```

```
## [1] 200 784
```

```r
# setting up an empty vector to later store the values of Feature #1 in
test_x1 <- vector(mode = "double", length = 200)

# running a for loop with our Feature #1 function to fill in the empty vector
# with the values of Feature #1 for the 200 observations in the testing data set
for (i in 1:200) {
  test_x1[i] <- get_center_left_border(testing_37_matrix[i,])
}

# setting up an empty vector to later store the values of Feature #2 in
test_x2 <- vector(mode = "double", length = 200)

# running a for loop with our Feature #2 function to fill in the empty vector
# with the values of Feature #2 for the 200 observations in the testing data set
for (i in 1:200) {
  test_x2[i] <- bottom_right_quadrant(testing_37_matrix[i,]) # filling the vector
}

# creating the official testing data set; there is a column for the digit (3 or 7),
# the value of Feature #1 (x_1), the value of Feature #2 (x_2), and the row
# number that a given observation corresponds to in the matrix that was stored
test_37_tbl <- tibble(y = as.factor(digit_37_test), x_1 = test_x1,
                      x_2 = test_x2, mat_row_num = 1:200)
```

```r
# the training data set
train_37_tbl
```

```
## # A tibble: 800 x 4
##    y       x_1   x_2 mat_row_num
##    <fct> <dbl> <dbl>       <int>
##  1 3     16.7   3947           1
##  2 3      7.33  9650           2
##  3 3     10.7   7149           3
##  4 7     13.3   4591           4
##  5 7     12.7   4615           5
##  6 7     10     6825           6
##  7 7      7.33 10050           7
##  8 7     15.7   4554           8
##  9 7      5     4179           9
## 10 3     14.3   6602          10
## # i 790 more rows
```

```r
# the testing data set
test_37_tbl
```

```
## # A tibble: 200 x 4
##    y       x_1   x_2 mat_row_num
##    <fct> <dbl> <dbl>       <int>
##  1 7     14.7   2107           1
##  2 7     15.7   8317           2
##  3 7      5    10280           3
##  4 3     12.3   6014           4
```

```
##  5 7     15.7   5880          5
##  6 3      9     6996          6
##  7 7      5.67  8674          7
##  8 3     10.7   8753          8
##  9 3      9     8728          9
## 10 7     17.3   7037         10
## # i 190 more rows
```

The training and testing data sets have now been created. We have two important matrices, `train_37_mat` and `test_37_mat`, as well as two important tables, `train_37_tbl` and `test_37_tbl`. The matrix `train_37_mat` contains the images of the 800 selected training digits with each image stored as a row of length 784; `test_37_mat` is the same but contains the 200 selected testing images instead. Information about these images is stored in the previously mentioned tables, `test_37_tbl` and `train_37_tbl`, both of which have the same format. These tables are the training and testing data sets, and the column values for them are: `y`, the digit class (either 3 or 7), `x_1`, the value of Feature #1, `x_2`, the value of Feature #2, and `mat_row_num`, the corresponding row value of the image in either `train_37_mat` or `test_37_mat`.

Model Creation, Optimization and Selection

The first model that we created is a KNN model. Since the KNN model has the parameter `k`, we need to determine what value of `k` is optimal. We do this by first creating a function, `calc_error`, which takes a `k` value, a training data set, and a testing data set, and returns the misclassification error for a KNN model with the specified `k` value. We ran this function using our training and testing data sets as inputs as well as `k` values from 1-100 to find the optimal `k` value, which ended up being 77.

```r
# CREATING A KNN MODEL AND CALCULATING THE MISCLASSIFICATION ERROR

# first line: setting up a function with inputs "k" and training + testing data sets
# second line: building a knn model based on a given value of the parameter "k"
# third line: storing the predicted values (3 or 7) of the model on the testing data
# fourth line: calculating the misclassification error for the model
calc_error <- function (kNear, train, test) {
  knn_model <- knn3(y ~ x_1 + x_2, data = train, k = kNear)
  pred <- predict(knn_model, test, type = "class")
  mean(pred != test$y)
}

# setting up an empty vector to later store the misclassification errors for the
# knn model using values of k from 1 to 100
mis_error <- vector(mode = "integer", length = 100)

# running a for loop to fill the empty vector with the misclassification errors
for (i in 1:100) {
  mis_error[i] <- calc_error(i, train_37_tbl, test_37_tbl)
}

# first line: making a tibble with a column for the error and one for the value of k
# second line: filtering for the value of k that produces the smallest error
tibble(mis_error = mis_error, k = 1:100) %>%   # optimal k = 77, error = 0.405
  slice_min(mis_error)
```

```
## # A tibble: 1 x 2
##   mis_error     k
##       <dbl> <int>
## 1     0.405    77
```

```
# from the above command, the lowest misclassification error is 0.405, which
# corresponds to an optimal k value of 77

# building the knn model using the optimal value of k (k = 77)
knn_model <- knn3(y ~ x_1 + x_2, data = train_37_tbl, k = 77)
```

The second model that we created is a logistic regression model. The logistic model has no parameters, so there is no need to calculate any optimal value. For this logistic regression model, we have created the recipe, built the model, defined the workflow, and then fit the workflow to our testing data set. We then add the predictions from the model onto the testing table and use those to determine the misclassification error for the model.

```
# CREATING A LOGISTIC REGRESSION MODEL AND CALCULATING THE MISCLASSIFICATION ERROR

# creating a recipe to define the role of the variables in our model
recipe_37 <- recipe(y ~ x_1 + x_2, data = train_37_tbl)

# creating our model and deciding on the implementation
logit_model <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

# creating a workflow by combining the recipe and our model
logit_wflow <- workflow() %>%
  add_recipe(recipe_37) %>%
  add_model(logit_model)

# fitting the workflow using our training data set
logit_fit <- fit(logit_wflow, train_37_tbl)

# augmenting the predictions of the model for the testing data set onto the
# testing data set
test_37_tbl <- augment(logit_fit, test_37_tbl)

# calculating the misclassification error for our model
# (which is equivalent to 1 minus our model's accuracy)
1 - (accuracy(data = test_37_tbl, truth = y, estimate = .pred_class)$.estimate)
```

```
## [1] 0.39
```

```
# the logistic regression model's misclassification error is 0.39
```

For our KNN model, using the optimal `k` value that we determined to be 77, the misclassification rate was 0.405. For our logistic model, the misclassification rate was 0.39. Therefore, we will be using the logistic model going forward, as it is has a slightly lower error rate than the KNN model. By creating a confusion matrix for this model (as seen in the code below), we see that the model is better at correctly predicting 3s than it is at correctly predicting 7s; the misclassification rate for 7s is about 42.2%, while for 3s, it is only about 35.9%.

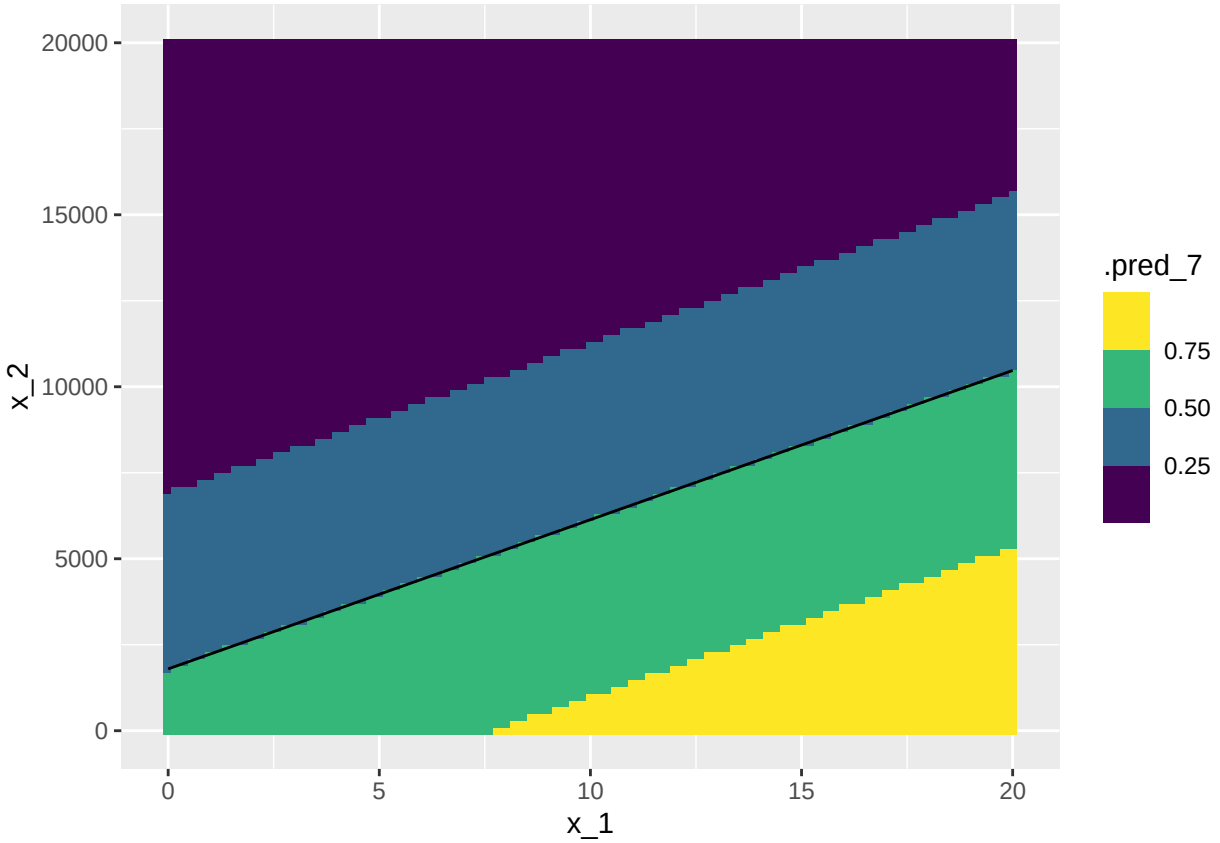
```r
# SELECTING ONE OF THE MODELS AND CALCULATING ITS CONFUSION MATRIX

# misclassification error for the knn model: 0.405
# misclassification error for the logistic regression model: 0.39

# thus, the logistic regression model has a lower misclassification error

# calculating the confusion matrix for the logistic regression model
conf_mat(data = test_37_tbl, truth = y, estimate = .pred_class)
```

```
##            Truth
## Prediction  3  7
##          3 66 41
##          7 37 56
```

Visualization

In order to see more closely how the average left center border (represented by variable `x_1`) and the summed intensity of the pixels in the bottom right quadrant (represent by variable `x_2`) affect the classification process, let's visualize the probabilities across a grid and locate the decision boundary for the model. In order to do this, we create a sequence of values in the range of Feature #1 and a sequence of values in the range of Feature #2, and then we create a table whose rows denote all of the possible combinations of the values between those two sequences.

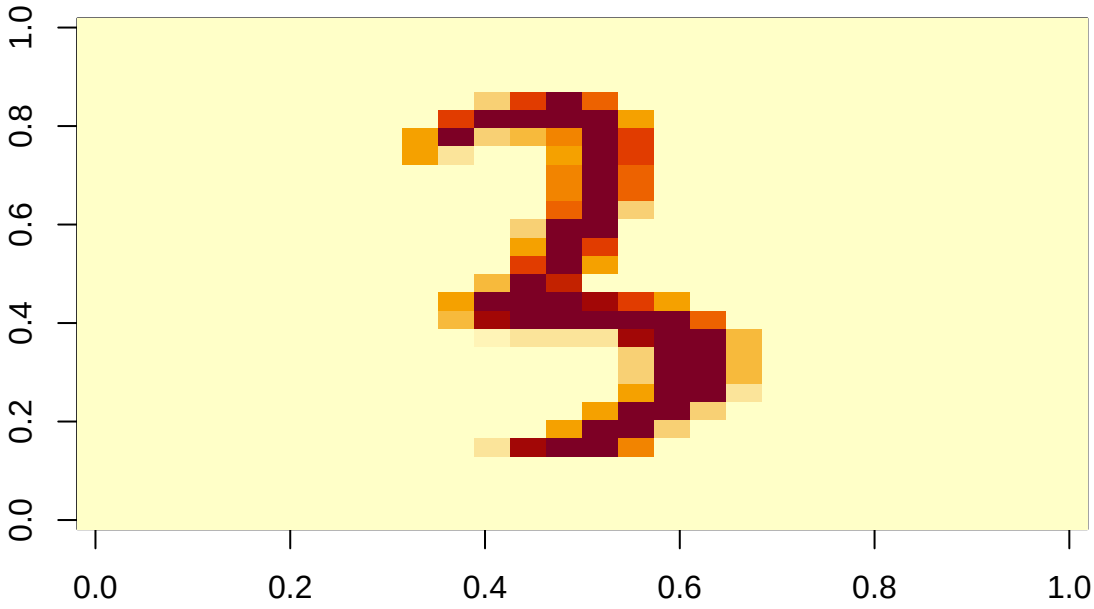
```r
# PLOTTING THE PROBABILITIES + DECISION BOUNDARY FOR OUR SELECTED MODEL

# creating a 100-length vector that takes on values from 0 to 20, which is the
# general range of values for Feature #1
grid_vec1 <- seq(0, 20, by = 0.2)

# creating a 100-length vector that takes on values from 0 to 20,000, which is the
# general range of values for Feature #2
grid_vec2 <- seq(0, 20000, by = 200)

# creating a tibble with every combination of values from the two vectors
grid_tbl <- expand_grid(x_1 = grid_vec1, x_2 = grid_vec2)

# using the created tibble as a testing data set and augmenting the predictions
# of the model onto that testing data set
grid_tbl <- augment(logit_fit, grid_tbl)

# plotting the probability that a digit is a 7 across a grid and adding a
# decision boundary at probability = 0.5
ggplot(grid_tbl, aes(x_1, x_2, z = .pred_7, fill = .pred_7)) +
  geom_raster() +
  stat_contour(breaks = c(0.5), color = "black") +
  scale_fill_viridis_b()
```

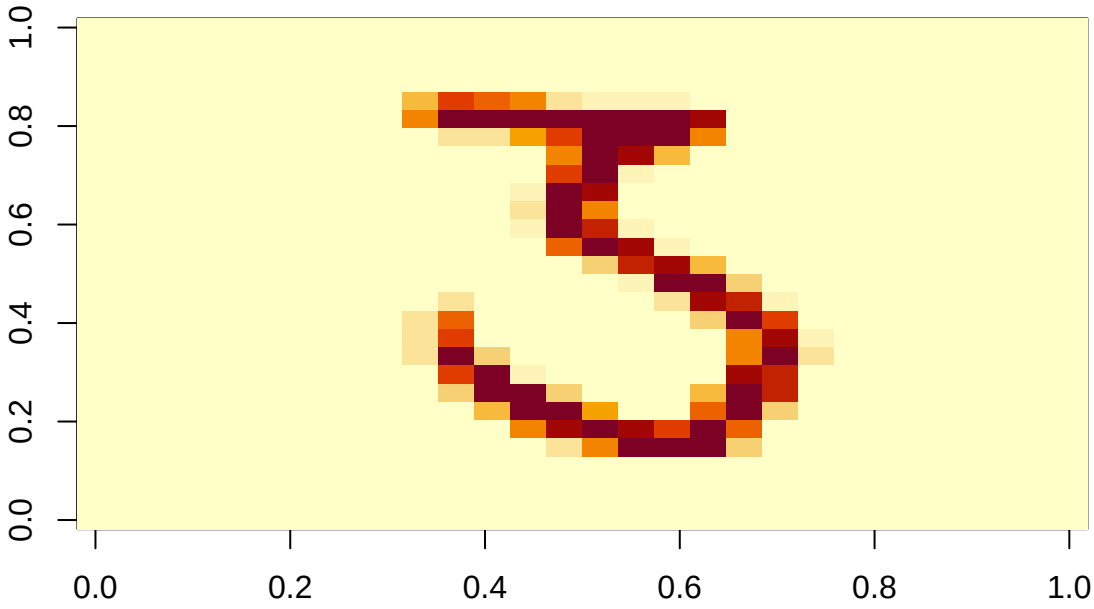


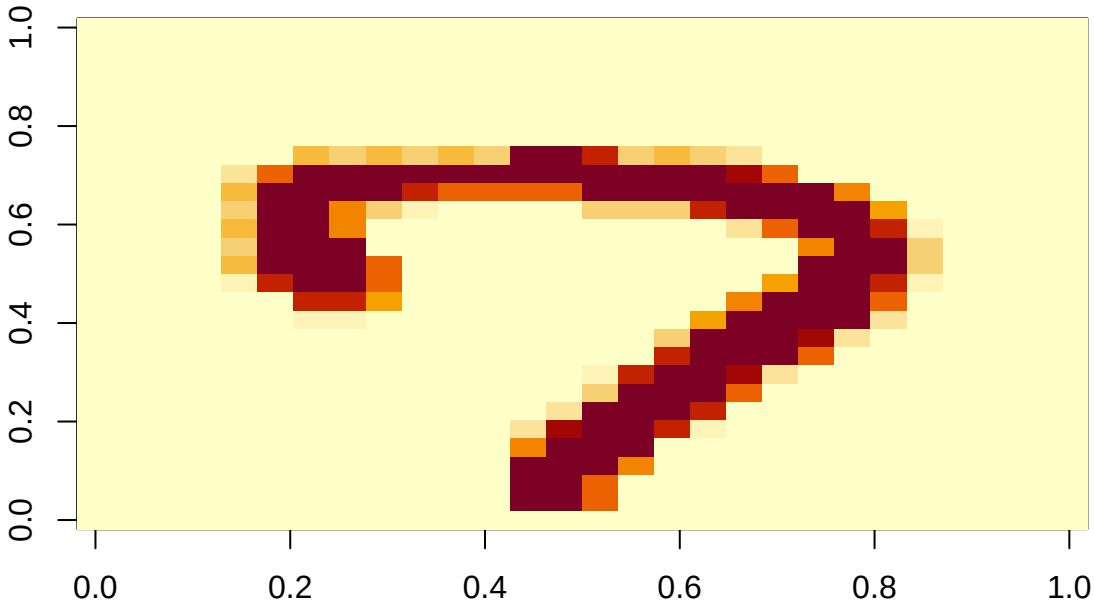
The plot displays the probability that a digit is a 7, and so probabilities less than 0.50 (above the black line decision boundary) mean that a digit will be classified as a 3. From the plot, as the average left center border gets bigger and the summed intensity in the bottom right quadrant gets smaller, the probability of that image being classified as a 7 increases. The decision boundary at 50% indicates that images that have a larger summed pixel intensity in the bottom right quadrant are more likely to be classified as 3s.

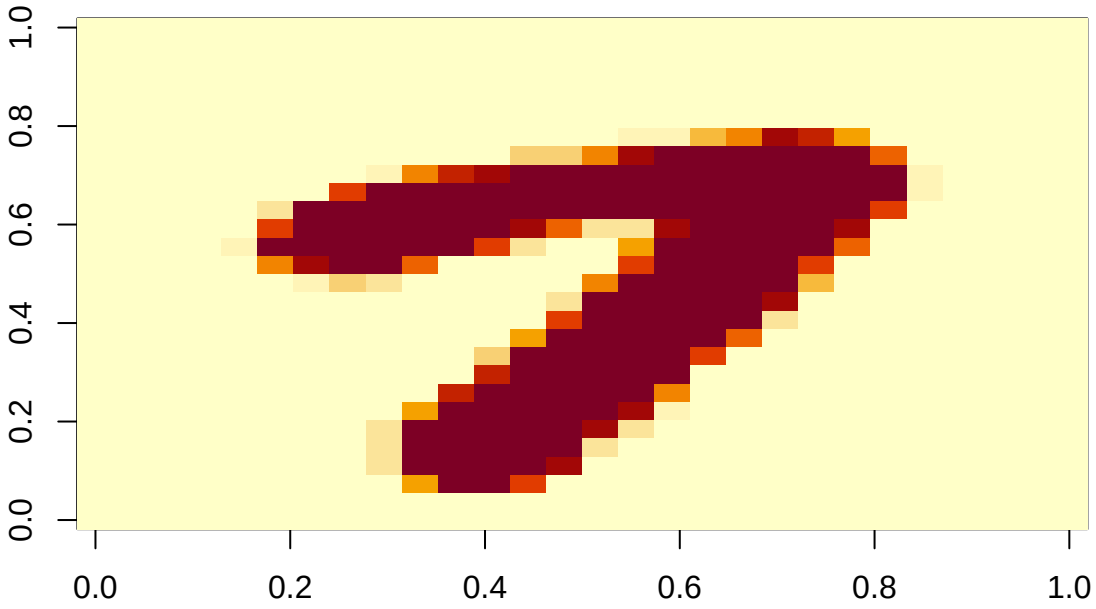
```
# FINDING TWO MISCLASSIFIED 3s AND TWO MISCLASSIFIED 7s

# filtering the testing data set to find two 3s that have been misclassified as
# 7s and two 7s that have been misclassified as 3s
test_37_tbl %>%
  filter(y != .pred_class) %>%
  slice_head(n = 4)
```

```
## # A tibble: 4 x 7
##   y       x_1   x_2 mat_row_num .pred_class .pred_3 .pred_7
##   <fct> <dbl> <dbl>       <int> <fct>         <dbl>   <dbl>
## 1 7      5    10280           3 3             0.795   0.205
## 2 3     12.3   6014           4 7             0.440   0.560
## 3 7      5.67  8674           7 3             0.720   0.280
## 4 3     14.7   6036          14 7             0.388   0.612
```


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0



From the above table, the two misclassified 3s are misclassified because they have a smaller number of non-zero pixels in the bottom right quadrant (and thus a lower summed intensity, which from the probability grid is typically associated more with 7s). For the first plotted misclassified 3, its summed intensity is only 6,014, and for the second plotted misclassified 3, its summed intensity is only 6,036. Also from the above table, the first plotted misclassified 7 has an unexpectedly low average left center border (5), as does the second plotted misclassified 7 (average left center border = 5.67). From the probability grid, we know that 7s tend to have higher average left center borders than 3s.

Changing things up (I)

In this section, we are rebuilding our training and testing data sets similarly to how we built them initially, with the added step of running the `contrast_digit` function on each row/image in both the training and testing matrices before calculating Feature #1 and Feature #2. These contrasted images are stored in `training_37_matrix_c` and `testing_37_matrix_c`. The new Feature #1 (`x_1`) and Feature #2 (`x_2`) values are stored in `train_37_tbl_c` and `test_37_tbl_c`, which will act as the new training and testing data sets for our model. The code to build these new data sets has been purposefully left out due to its extreme similarity to the code used to build the initial training and testing data sets.

```
# the contrasted training data set
train_37_tbl_c
```

```
## # A tibble: 800 x 4
##    y       x_1   x_2 mat_row_num
```

```
##    <fct> <dbl> <dbl>       <int>
##  1 3     17.7   3315           1
##  2 3      8.33  9690           2
##  3 3     11.3   6375           3
##  4 7     14.3   4590           4
##  5 7     15     4590           5
##  6 7     11     7140           6
##  7 7      8.33 10455           7
##  8 7     16.3   4335           8
##  9 7      5.33  3315           9
## 10 3     15     7140          10
## # i 790 more rows
```

```
# the contrasted testing data set
test_37_tbl_c
```

```
## # A tibble: 200 x 4
##    y       x_1   x_2 mat_row_num
##    <fct> <dbl> <dbl>       <int>
##  1 7     15     1275           1
##  2 7     16.3   7905           2
##  3 7      6    10710           3
##  4 3     13.3   5355           4
##  5 7     16.7   5865           5
##  6 3     10     7140           6
##  7 7      9.33  8925           7
##  8 3     12     8925           8
##  9 3     10.3   8415           9
## 10 7     18.3   6885          10
## # i 190 more rows
```

After the new data sets have been built with the correct alterations to the image vectors, the logistic model that we selected earlier can be rebuilt, in the same way that it was the first time but with the new training data set. Nothing about this process is different, since the only thing we changed was the initial images used, not any features or other aspects of the model.

```
# RETRAINING THE LOGISTIC MODEL AND CALCULATING THE NEW MISCLASSIFICATION ERROR

# creating a recipe to define the role of the variables in our model
recipe_37_c <- recipe(y ~ x_1 + x_2, data = train_37_tbl_c)

# creating our model and deciding on the implementation
logit_model_c <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

# creating a workflow by combining the recipe and our model
logit_wflow_c <- workflow() %>%
  add_recipe(recipe_37_c) %>%
  add_model(logit_model_c)

# fitting the workflow using our training data set
logit_fit_c <- fit(logit_wflow_c, train_37_tbl_c)
```

```
# augmenting the predictions of the model for the testing data set onto the
# testing data set
test_37_tbl_c <- augment(logit_fit_c, test_37_tbl_c)

# calculating the misclassification error for our model
# (which is equivalent to 1 minus our model's accuracy)
1 - (accuracy(data = test_37_tbl_c, truth = y, estimate = .pred_class)$.estimate)
```

```
## [1] 0.385
```

```
# the contrasted logistic regression model's misclassification error is 0.385
```

Our misclassification error rate for the contrasted model is 38.5%, which is just slightly better than the 39% misclassification error rate of the the first logistic model, athough the two values are essentially the same.

Changing things up (II)

Building a classifier to distinguish between 2 digits seems interesting but lacks practicality. In real life, there can be situations in which there are more than 2 different digits that need to be classified. Therefore, adding at random a new digit to the data sets and re-applying the model makes the algorithm more applicable and relevant to more advanced situations. In order to add the 6s to the training and testing data sets, training and testing data sets were created with just the 400 and 100 6s respectively (through the same process that the initial training and testing data sets were created by), and then the just-6s training and testing data sets were binded to the initial training and testing data sets to create ones that had 3s, 6s, and 7s. The code to build these new data sets has been purposefully left out due to its extreme similarity to the code used to build the initial training and testing data sets. The model creation, misclassification error rate calculation, and confusion matrix creation were done in the exact same way as with the first logistic model; only the training and testing data sets have been altered. The testing data set for the probability grid is also identical to the one used before, but now, digits can be classified into three possible classes rather than just 2, and so the plot displays regions of feature values that correspond to a certain class rather than the probability that a digit is a 7.

```
# the training data set with 3s, 6s, and 7s
train_367_tbl
```

```
## # A tibble: 1,200 x 4
##    y       x_1   x_2 mat_row_num
##    <fct> <dbl> <dbl>       <int>
##  1 3     16.7   3947           1
##  2 3      7.33  9650           2
##  3 3     10.7   7149           3
##  4 7     13.3   4591           4
##  5 7     12.7   4615           5
##  6 7     10     6825           6
##  7 7      7.33 10050           7
##  8 7     15.7   4554           8
##  9 7      5     4179           9
## 10 3     14.3   6602          10
## # i 1,190 more rows
```

```r
# the testing data set with 3s, 6s, and 7s
test_367_tbl
```

```
## # A tibble: 300 x 4
##    y       x_1   x_2 mat_row_num
##    <fct> <dbl> <dbl>       <int>
##  1 7     14.7   2107           1
##  2 7     15.7   8317           2
##  3 7      5    10280           3
##  4 3     12.3   6014           4
##  5 7     15.7   5880           5
##  6 3      9     6996           6
##  7 7      5.67  8674           7
##  8 3     10.7   8753           8
##  9 3      9     8728           9
## 10 7     17.3   7037          10
## # i 290 more rows
```

```r
# RETRAINING THE LOGISTIC MODEL AND CALCULATING THE NEW MISCLASSIFICATION ERROR

# defining the factor levels for the new training data set
train_367_tbl <- train_367_tbl %>%
  mutate(y = fct_relevel(y, c("3","6","7")))

# defining the factor levels for the new testing data set
test_367_tbl <- test_367_tbl %>%
  mutate(y = fct_relevel(y, c("3","6","7")))

# loading the library that allows us to do multinomial logistic regression
library("glmnet")

# creating our model and deciding on the implementation
logit_model_367 <- multinom_reg(mode = "classification", engine = "nnet")

# creating a recipe to define the role of the variables in our model
recipe_367 <- recipe(y ~ x_1 + x_2, data = train_367_tbl)

# creating a workflow by combining the recipe and our model
logit_wflow_367 <- workflow() %>%
  add_recipe(recipe_367) %>%
  add_model(logit_model_367)

# fitting the workflow using our training data set
logit_fit_367 <- fit(logit_wflow_367, train_367_tbl)

# augmenting the predictions and probabilities of the model for the testing data
# set onto the testing data set
test_367_tbl <- augment(logit_fit_367, test_367_tbl)

# calculating the misclassification error for our model
# (which is equivalent to 1 minus our model's accuracy)
1 - (accuracy(data = test_367_tbl, truth = y, estimate = .pred_class)$.estimate)
```

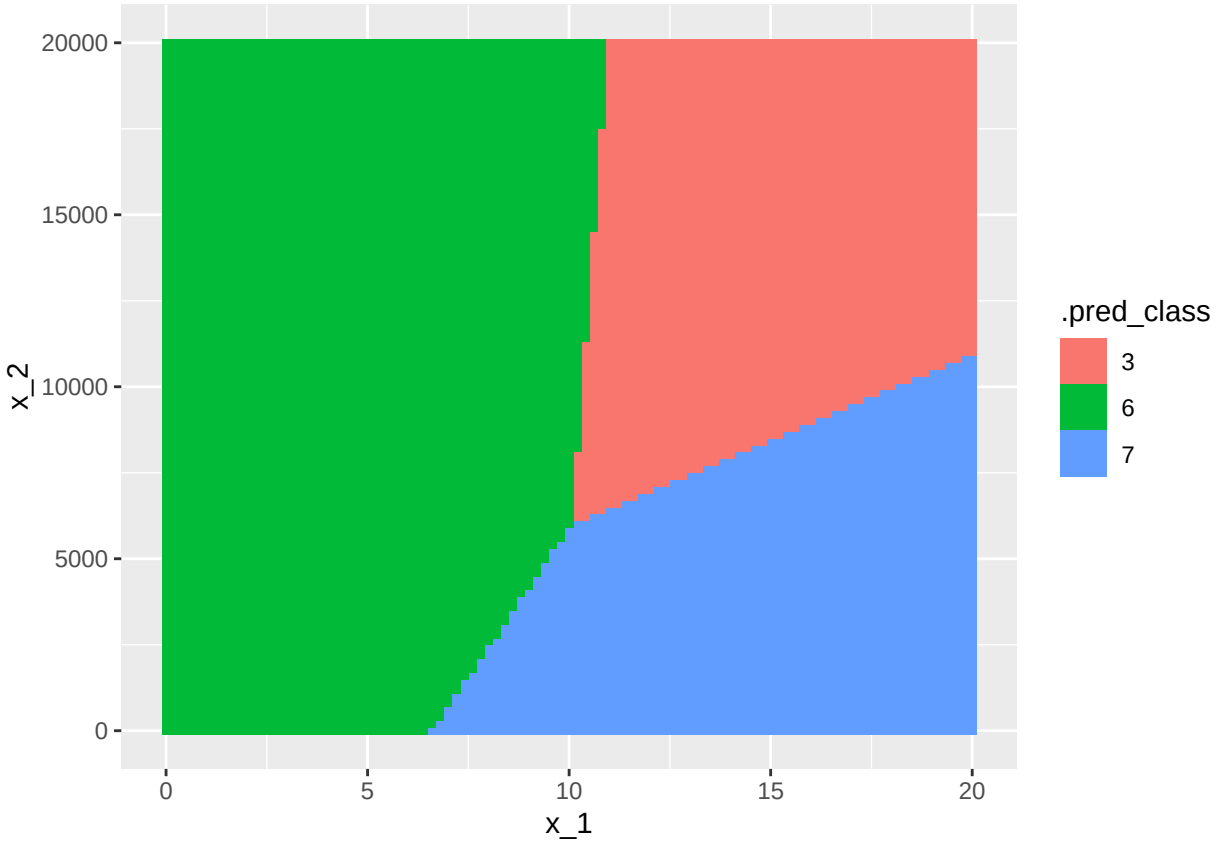
```
## [1] 0.44
```

```
# the new logistic regression model's misclassification error is 0.44
```

```
# CALCULATING THE CONFUSION MATRIX FOR THE NEW LOGISTIC REGRESSION MODEL

conf_mat(data = test_367_tbl, truth = y, estimate = .pred_class)
```

```
##           Truth
## Prediction  3  6  7
##          3 45 14 17
##          6 22 72 29
##          7 36 14 51
```

```
# PLOTTING THE PROBABILITIES + DECISION BOUNDARY FOR THE MULTINOMIAL MODEL

# creating a tibble with every combination of Feature #1 and Feature #2
grid_tbl_1 <- expand_grid(x_1 = grid_vec1, x_2 = grid_vec2)

# using the created tibble as a testing data set and augmenting the predictions
# of the model onto that testing data set
grid_tbl_1 <- augment(logit_fit_367, grid_tbl_1)

# plotting the probability of a given digit
grid_tbl_1 %>%
  ggplot(mapping = aes(x = x_1, y = x_2, fill = .pred_class)) +
    geom_raster()
```



The rate of getting predicted correctly for 3s, 6s, 7s are 44%, 72%, and 53% respectively, which means that 3s are the most confusing digit among the three. The reason for the high misclassification rate for 3s is that some 3s have unexpectedly low summed pixel intensity in the bottom right quadrant, which makes it easy for them to get misclassified with 7s (which have lower summed intensity in that area). There are 22 out of 103 3s that are recognized as 6s, which likely occurs because of a similarity in the average left center border between 3s and 6s.