Project

Saji, Joud, Markos

3/16/2022

Introduction

In class we used classifier to distinguish between handwritten 2 and 7. In the dataset we had predefined features and used them straight ahead. For this project we will be defining our own features to create an algorithm to distinguish between 5 and 8. We will be using the same dataset mnist to get the images and labels for those numbers. In the beggining we create training and testing datasets, filter images that have label 5 or 8 and put it in the dataset. Additionally, we sampled 800 rows for training dataset and 200 rows fo testing dataset. We use a seed to make our results reproducible.

```
set.seed(987654321)

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

mnist58.train <- tibble(images = mnist$train$images,
                        labels = mnist$train$labels) %>%
                 dplyr::filter(labels == 5 | labels == 8) %>%
                 mutate(labels = as.factor(labels)) %>%
                 sample_n(800)
mnist58.train
```

```
## # A tibble: 800 x 2
##    images[,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11] labels
##         <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <fct>
##  1          0     0     0     0     0     0     0     0     0     0     0 5
##  2          0     0     0     0     0     0     0     0     0     0     0 5
##  3          0     0     0     0     0     0     0     0     0     0     0 8
##  4          0     0     0     0     0     0     0     0     0     0     0 8
##  5          0     0     0     0     0     0     0     0     0     0     0 8
##  6          0     0     0     0     0     0     0     0     0     0     0 8
##  7          0     0     0     0     0     0     0     0     0     0     0 5
##  8          0     0     0     0     0     0     0     0     0     0     0 8
##  9          0     0     0     0     0     0     0     0     0     0     0 8
## 10          0     0     0     0     0     0     0     0     0     0     0 5
## # ... with 790 more rows, and 1 more variable: images[12:784] <int>
```

```
mnist58.test <- tibble(images = mnist$test$images,
                       labels = mnist$test$labels) %>%
                dplyr::filter(labels == 5 | labels == 8) %>%
                mutate(labels = as.factor(labels)) %>%
                sample_n(200)
mnist58.test
```

```
## # A tibble: 200 x 2
```

```
##    images[,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11] labels
##         <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <fct>
##  1          0     0     0     0     0     0     0     0     0     0     0 5
##  2          0     0     0     0     0     0     0     0     0     0     0 5
##  3          0     0     0     0     0     0     0     0     0     0     0 8
##  4          0     0     0     0     0     0     0     0     0     0     0 8
##  5          0     0     0     0     0     0     0     0     0     0     0 8
##  6          0     0     0     0     0     0     0     0     0     0     0 8
##  7          0     0     0     0     0     0     0     0     0     0     0 8
##  8          0     0     0     0     0     0     0     0     0     0     0 5
##  9          0     0     0     0     0     0     0     0     0     0     0 8
## 10          0     0     0     0     0     0     0     0     0     0     0 8
## # ... with 190 more rows, and 1 more variable: images[12:784] <int>
```

Defining and implementing features

We have tried several features related to the 4 quadrants of the image, however those did not end up with high accuracy rate. Instead we figured that for 8 there would be more concentration of black pixels in the very middle of the image. Because of that our first feature takes a 6 by 6 square right in the middle of the image and calculates the sum of the darkness of each pixel. For our second feature we figured that the diagonal line from bottom left and upper right corner could be a good one. For number 8 it should capture darker pixels compared to 5. We use for loops to campture those features. Additionally, we used a nested loop to itearate from 1 to 28 and capture [i,i] index of the matrix. The accumulator takes the sum of those indexes and captures the seoncd feature for each row.

```
for (i in 1:800){
    x_2_acc = 0
    matx <- matrix(mnist58.train$images[i, ], nrow = 28)
    for (j in 1:28) {
      x_2_acc = x_2_acc + matx[j,j]
    }
    mnist58.train$x_1[i] = sum(matx[12:17,12:17])
    mnist58.train$x_2[i] = x_2_acc
}
```

```
## Warning: Unknown or uninitialised column: 'x_1'.
```

```
## Warning: Unknown or uninitialised column: 'x_2'.
```

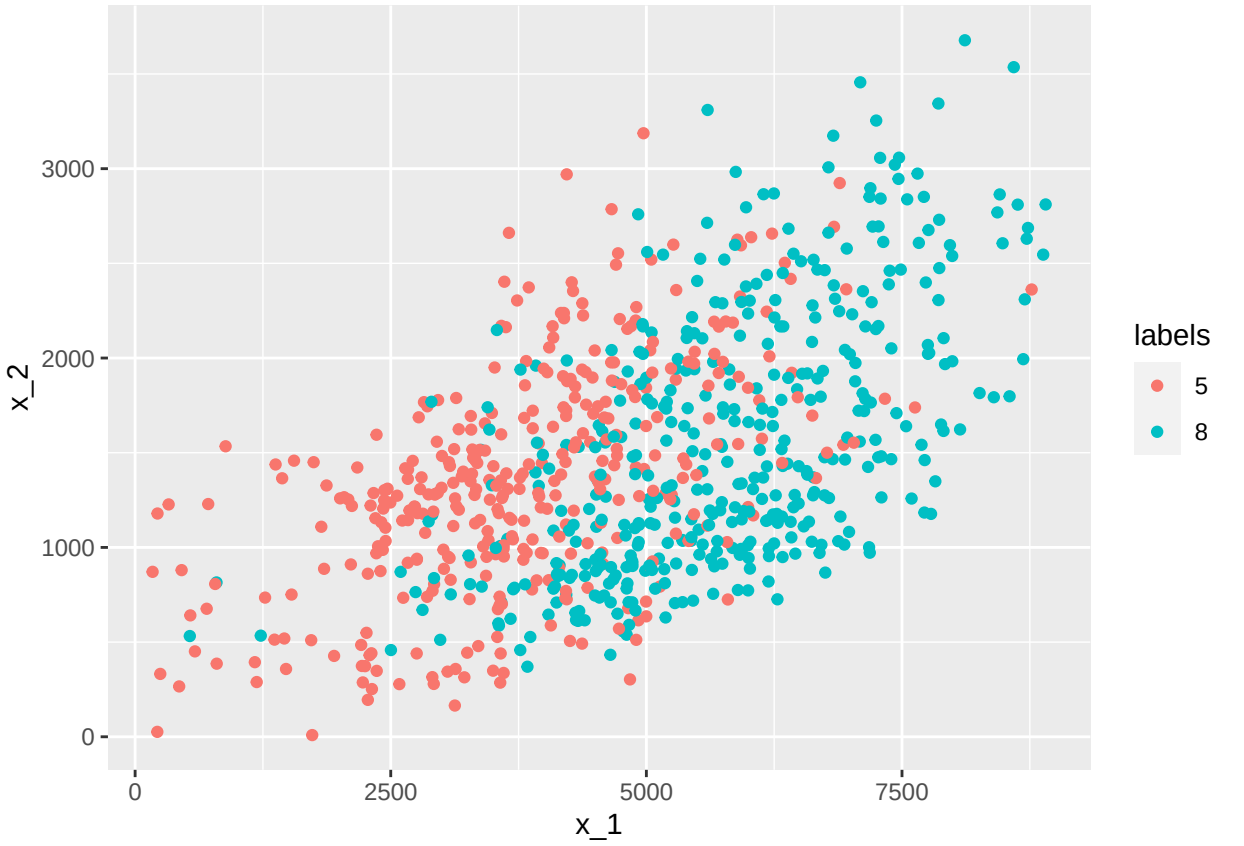
```
for (i in 1:200){
  x_2_acc = 0
  matx <- matrix(mnist58.test$images[i, ], nrow = 28)
  for (j in 1:28) {
    x_2_acc = x_2_acc + matx[j,j]
    }
    mnist58.test$x_1[i] = sum(matx[12:17,12:17])
    mnist58.test$x_2[i] = x_2_acc
}
```

```
## Warning: Unknown or uninitialised column: 'x_1'.
## Unknown or uninitialised column: 'x_2'.
```

```
mnist58.train <- mnist58.train %>%
  select(labels, x_1, x_2)
```

```
mnist58.test <- mnist58.test %>%
  select(labels, x_1, x_2)

mnist58.train %>%
  ggplot() +
  geom_point(aes(x_1, x_2, color = labels))
```



As we can see in the plot above the first ferature does a great job at separating red dots from the blue dots. We can also see that the second feature did not as well as predicted. We tried several others but this was the feature that resulted in higher accuracy.

Training and optimizing KNN model

For KDD we decided to create a for loop that will iterate through values of k from 1 to 50. It will have an accumulator that will store the accuracy rate and have a conditional statement inside the loop to keep the best model and the best accuracy.

```
best_k <- 0
best_model <- 0
best_acc <- 0

for (k in 1:50) {
  knn.model <- nearest_neighbor(neighbors = k) %>%
    set_engine("kknn") %>%
    set_mode("classification")
```

```
  mnist58.recipe <- recipe(labels ~ x_1 + x_2, data=mnist58.train)

  knn.wflow <- workflow() %>%
    add_recipe(mnist58.recipe) %>%
    add_model(knn.model)

  knn.fit <- fit(knn.wflow, mnist58.train)

  acc_tib <- augment(knn.fit, mnist58.test) %>%
    accuracy(truth = labels, estimate = .pred_class)

  acc <- acc_tib$.estimate

  if (acc > best_acc) {
    best_acc <- acc
    best_model <- knn.fit
    best_k <- k
  }
}

best_acc
```

```
## [1] 0.845
```

```
best_k
```

```
## [1] 31
```

```
knn.fit <- best_model
```

As we can see from the optimized KNN classification above we have an accuracy of 84.5%. In practice, it is not that high however in the context of this problem it seems like it is a good result and it would be fair to assume that the model along with features did well. We can try to do a logistic regression as well to see it that model performs better with our features.

Training logistic regression

Next, we are going to try is logistic regression. It is quite flexible and should perform well if the features are chosen correctly. We set up a model, recipe and workflow to get the fit and calculate the accuracy.

```
logit.model <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

mnist.recipe <- recipe(labels ~ x_1 + x_2, data=mnist58.train)

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

logit.fit <- fit(logit.wflow, mnist58.train)

augment(logit.fit, mnist58.test) %>%
  accuracy(truth = labels, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.865
```

As we can see above the model resulted in accuracy of 86.5% which seems good, in fact it is performed better then the KNN model. We can say that this worked well with the input variables that we choose.

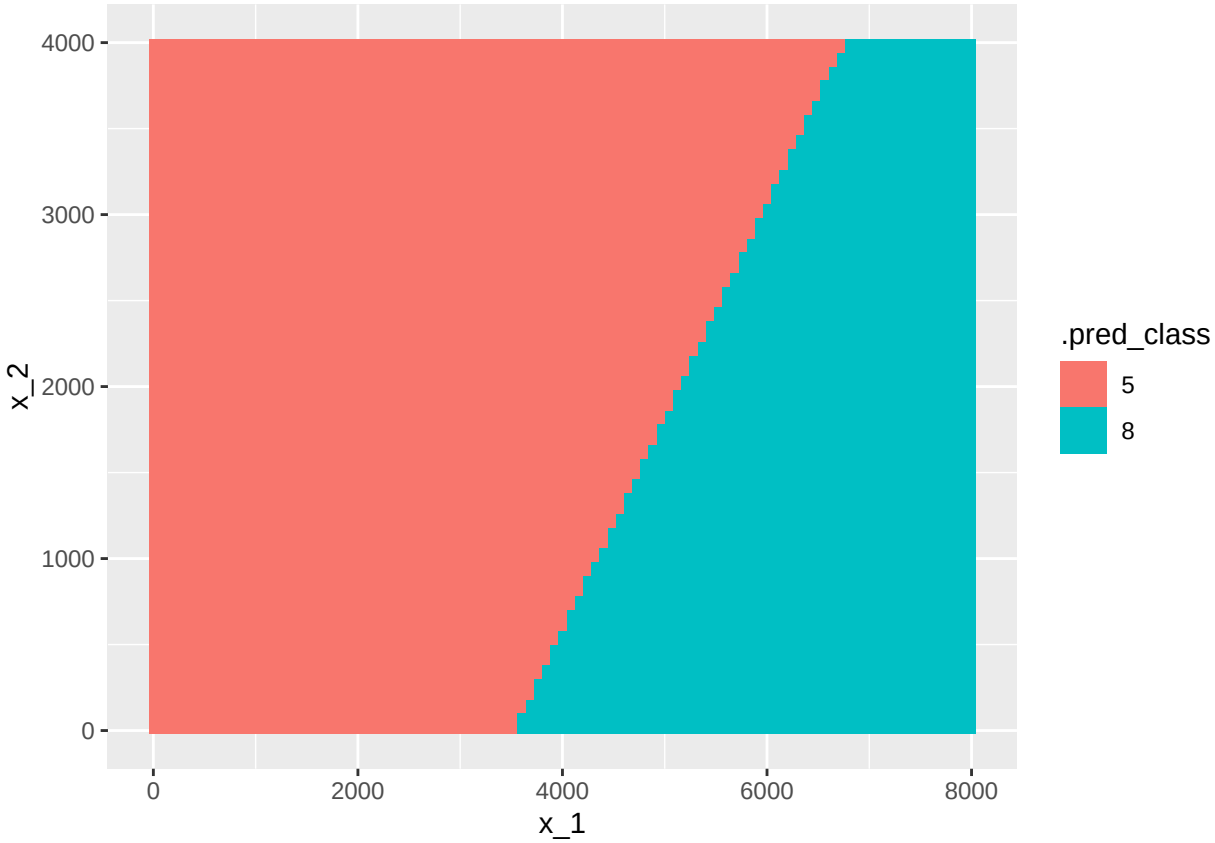
Decision boundary for the logistic regression

We can now create a grid tibble and plot the decision boundary using geom_raster. We made sure that the grid tibble does not have too much rows so it would not take a lost of space for the server to plot.

```
mnist58.grid <- expand_grid(x_1 = seq(0, 8000, 80),
                            x_2 = seq(0, 4000, 40))

mnist58.grid.new <- augment(logit.fit, mnist58.grid)

ggplot(mnist58.grid.new, aes(x = x_1, y = x_2, fill = .pred_class)) +
  geom_raster()
```



KNN for 5, 8 and 4

Currently we have a logistic regression that was created only considering the features for 5 and 8. However we need to remember that logistic regression is not fit for multi-class classification. Our KNN model performed quite well, so we can use it to classify this three digits.

```
set.seed(987654321)

mnist584.train <- tibble(images = mnist$train$images,
                         labels = mnist$train$labels) %>%
                  dplyr::filter(labels == 5 | labels == 8 | labels == 4) %>%
                  mutate(labels = as.factor(labels)) %>%
                  sample_n(800)
mnist584.train
```

```
## # A tibble: 800 x 2
##    images[,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11] labels
##         <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <fct>
##  1          0     0     0     0     0     0     0     0     0     0     0 5
##  2          0     0     0     0     0     0     0     0     0     0     0 4
##  3          0     0     0     0     0     0     0     0     0     0     0 5
##  4          0     0     0     0     0     0     0     0     0     0     0 4
##  5          0     0     0     0     0     0     0     0     0     0     0 8
##  6          0     0     0     0     0     0     0     0     0     0     0 8
##  7          0     0     0     0     0     0     0     0     0     0     0 4
##  8          0     0     0     0     0     0     0     0     0     0     0 5
##  9          0     0     0     0     0     0     0     0     0     0     0 4
## 10          0     0     0     0     0     0     0     0     0     0     0 4
## # ... with 790 more rows, and 1 more variable: images[12:784] <int>
```

```
mnist584.test <- tibble(images = mnist$test$images,
                        labels = mnist$test$labels) %>%
                 dplyr::filter(labels == 5 | labels == 8 | labels == 4) %>%
                 mutate(labels = as.factor(labels)) %>%
                 sample_n(200)
mnist584.test
```

```
## # A tibble: 200 x 2
##    images[,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11] labels
##         <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <fct>
##  1          0     0     0     0     0     0     0     0     0     0     0 4
##  2          0     0     0     0     0     0     0     0     0     0     0 5
##  3          0     0     0     0     0     0     0     0     0     0     0 8
##  4          0     0     0     0     0     0     0     0     0     0     0 8
##  5          0     0     0     0     0     0     0     0     0     0     0 4
##  6          0     0     0     0     0     0     0     0     0     0     0 8
##  7          0     0     0     0     0     0     0     0     0     0     0 4
##  8          0     0     0     0     0     0     0     0     0     0     0 5
##  9          0     0     0     0     0     0     0     0     0     0     0 4
## 10          0     0     0     0     0     0     0     0     0     0     0 5
## # ... with 190 more rows, and 1 more variable: images[12:784] <int>
```

```
for (i in 1:800){
    x_2_acc = 0
    matx <- matrix(mnist584.train$images[i, ], nrow = 28)
    for (j in 1:28) {
```

```
    x_2_acc = x_2_acc + matx[j,j]
  }
  mnist584.train$x_1[i] = sum(matx[12:17,12:17])
  mnist584.train$x_2[i] = x_2_acc
}
```

```
## Warning: Unknown or uninitialised column: `x_1`.
```

```
## Warning: Unknown or uninitialised column: `x_2`.
```

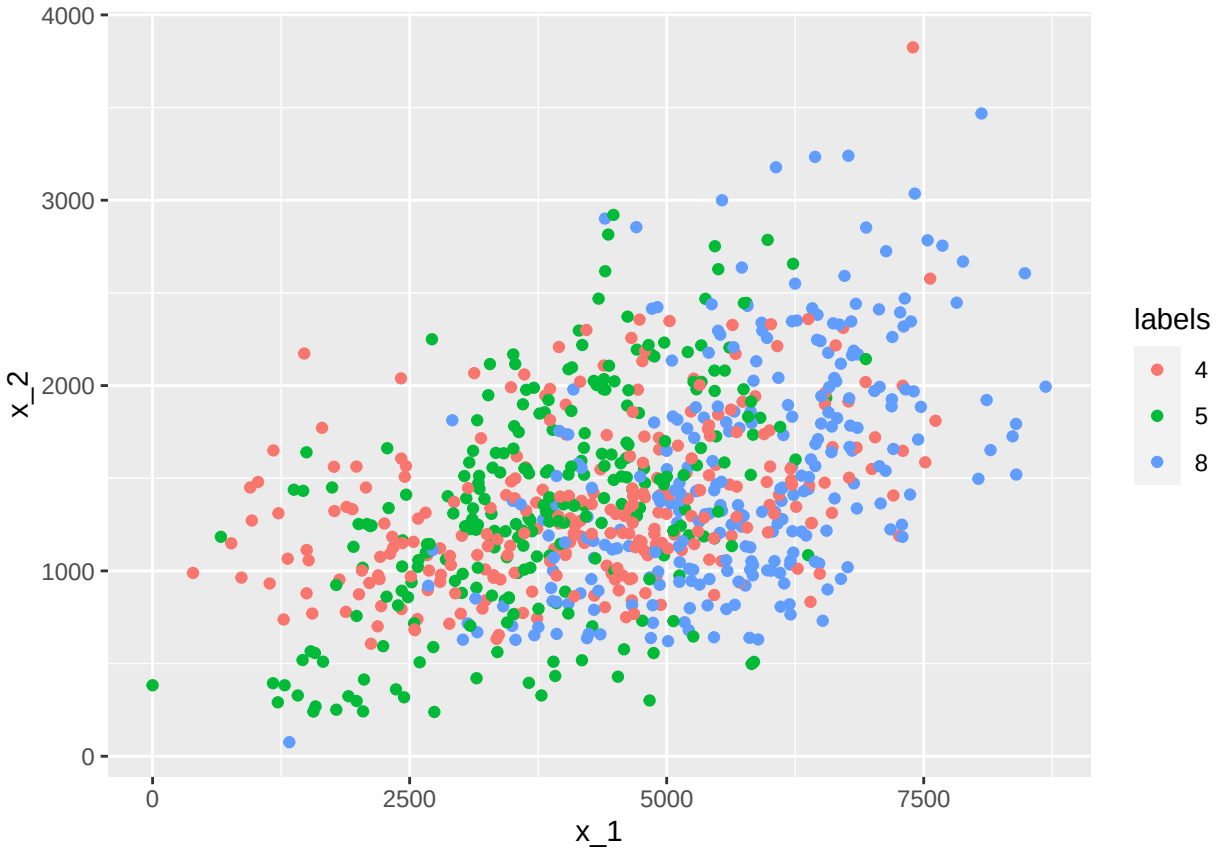
```
for (i in 1:200){
  x_2_acc = 0
  matx <- matrix(mnist584.test$images[i, ], nrow = 28)
  for (j in 1:28) {
    x_2_acc = x_2_acc + matx[j,j]
    }
  mnist584.test$x_1[i] = sum(matx[12:17,12:17])
  mnist584.test$x_2[i] = x_2_acc
}
```

```
## Warning: Unknown or uninitialised column: `x_1`.
## Unknown or uninitialised column: `x_2`.
```

```
mnist584.train <- mnist584.train %>%
  select(labels, x_1, x_2)

mnist584.test <- mnist584.test %>%
  select(labels, x_1, x_2)

mnist584.train %>%
  ggplot() +
  geom_point(aes(x_1, x_2, color = labels))
```



As we can see above all of the points seem to be messy and 4 is spread all over the x_1 and x_2 axis. We already can predict that the model will not do well on this 3 digits.

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

mnist584.recipe <- recipe(labels ~ x_1 + x_2, data=mnist584.train)

knn.wflow <- workflow() %>%
  add_recipe(mnist584.recipe) %>%
  add_model(knn.model)

knn.fit <- fit(knn.wflow, mnist584.train)

acc_tib <- augment(knn.fit, mnist584.test) %>%
  accuracy(truth = labels, estimate = .pred_class)

acc <- acc_tib$.estimate

acc
```

```
## [1] 0.61
```

```
augment(knn.fit, mnist584.test) %>%
  accuracy(truth = labels, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
```

```
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass      0.61
```

```
augment(knn.fit, mnist584.test) %>%
  conf_mat(truth = labels, estimate = .pred_class)
```

```
##            Truth
## Prediction  4  5  8
##          4 31  9  7
##          5 17 37 11
##          8 21 13 54
```

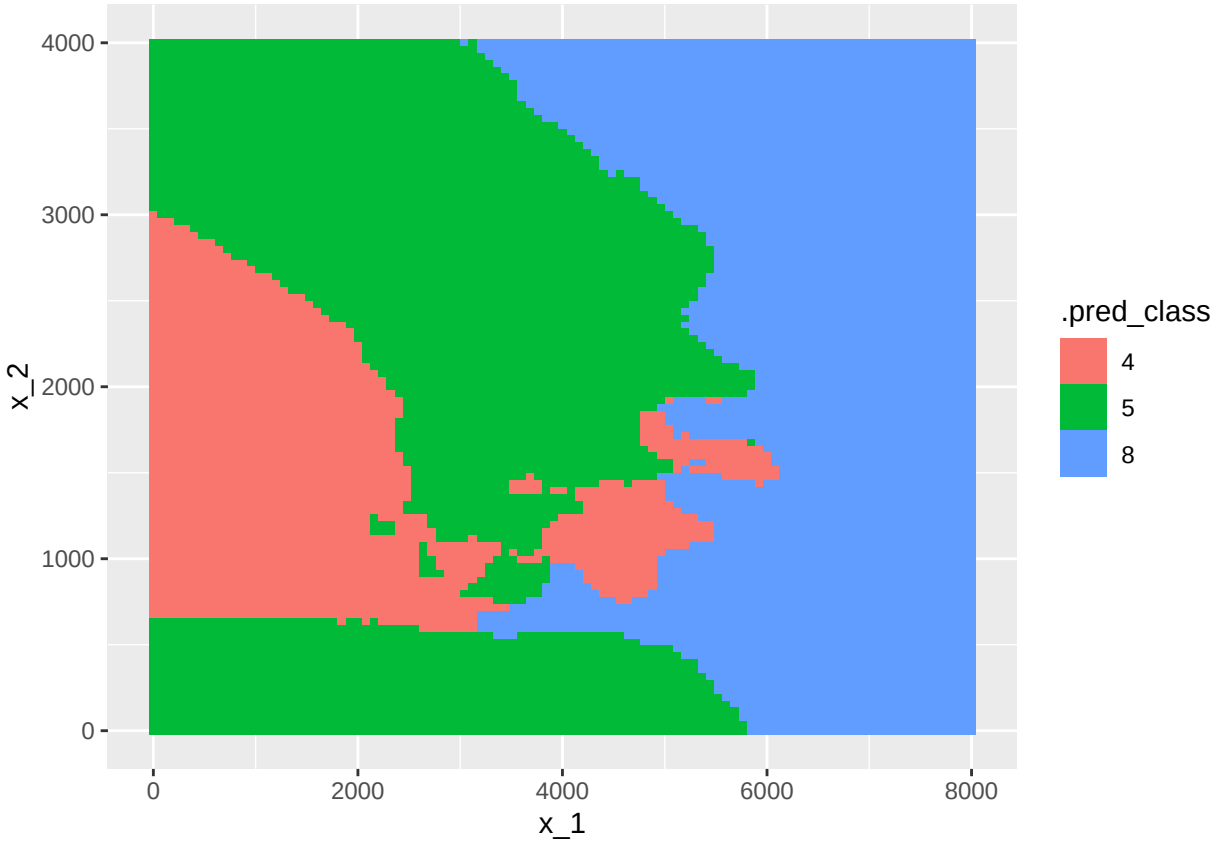
As we can see above the accuracy of the knn model, with the optimal k that we determined in the model that included 5 and 8 only, is 52%. As we expected this result is very low. On the same note, if we look at the confusion matrix we can see that 5 and 8 got misclassified not that much. In fact it did a good job for those two numbers, especially 8. However we can see that 4 got misclassified a lot. This two features that we chose do not work well with 4.

Decision boundary for the KNN model of three digits

```
mnist584.grid <- expand_grid(x_1 = seq(0, 8000, 80),
                             x_2 = seq(0, 4000, 40))

mnist584.grid.new <- augment(knn.fit, mnist584.grid)

ggplot(mnist584.grid.new, aes(x = x_1, y = x_2, fill = .pred_class)) +
  geom_raster()
```



As we can see above the decision boundary does not look good as well. We can see that it is curved a lot indicating that the features did not work really well with the set of this 3 numbers.

Conclusion

During our project we developed a logistic regression and KNN models that used two input features to determine weather the digit in the image is 5 or 8. Additionally, we chose the input variables by ourselves. The first feature was the sum of the 6 by 6 square matrix right in the center of the image. The second feature was the sum of diagonal indices of the matrix of the image (from lower left corner to upper right corner). With KNN model we achieved 84.5% accuracy which is decently good. For logistic regression we got an accuracy of 86.5%. We also plotted the decision boundary for the logistic regression. Afterwards we implemented the KNN algorithm on digits 5, 8 and 4 to check how the model not created for digit 4 will perform. Logistic regression performed better then KNN however it is not fit for multi-class classification, so we used KNN for this task. We saw that the accuracy of this model was 52%, which is very low. We could also see from the decision boundary that the model did not perform well as the boundaries were curved a lot.