Challange 1

Noah Johnson, Bryce Gmyrek, Matthew Myers, Andrew Noecker

3/16/2022

Feature Definition

First Feature: differences in pixel averages

The first feature is defined as follows. First, filter the training data set to create two separate data sets including only 2s and 3s respectively. Then, for each of the 784 pixels, calculate the average darkness for each pixel, creating a new matrix. Next, subtract the 2 matrix from the 3 matrix, to yield a matrix showing the differences between 2s and 3s for each pixel which, for simplicity, we convert this matrix into a vector of length 728. To create the feature, we convert each of the images in the train data set into vectors of length 728, with each entry representing a pixel. Taking the dot product of the two vectors yields a score with a higher value suggesting a 2 and a lower value suggesting a 3. Intuitively, this approach could be defined as looking at the difference between a given image and the average image by each cell using an average weighted by each cell's importance. We chose this feature because we hypothesized that it would easily differentiate numbers due to the fact that it considers each pixel but weighs to focus on the most important ones.

```r
#feature 1 involving the pixel average
pixel.means = data.frame(matrix(nrow = 784,ncol = 2))
indicator.2 = mnist$train$labels==(2)
indicator.3 = mnist$train$labels==(3)

# iterating through pixels to find the average darkness for 2's and 3's
for (i in 1:784){
  pixel <- tibble(
    darkness = mnist$train$images[,i],
    indicator.2,
    indicator.3
  )

  Mean.2 <- pixel %>%
    filter(indicator.2 == TRUE)%>%
    summarize(mean = mean(darkness))

  Mean.3 <- pixel %>%
    filter(indicator.3 == TRUE)%>%
    summarize(mean = mean(darkness))

  pixel.means[1][i,] = as.numeric(Mean.2)
  pixel.means[2][i,] = as.numeric(Mean.3)
}

# difference in average darkness by pixel between 2's and 3's
```

```
mean.diff <- pixel.means%>%
  mutate(diff = X1 - X2)

index_235 <- which(mnist$train$labels %in% c(2,3,5))
y <- mnist$train$labels[index_235]
x <- mnist$train$images[index_235,]

# Calculating feature 1 for all 2's and 3's, and 5's
n = length(y)
feature.1.vec = vector(length = n)

for (i in 1:n){
  feature.1.vec[i]<- mean.diff %>%
    mutate(darkness = x[i,],
           product = diff*darkness)%>%
    summarize(sum = sum(product))%>%
    as.numeric()
}

f1.tbl = tibble(
  letter = as.factor(y[1:n]),
  feature.1 = feature.1.vec
)
```

Second Feature: top-bottom symmetry of amount of ink

Our second feature is counting the number of pixels that are inked and finding what proportion of those are in the top vs bottom half of the image, and then taking the difference in those proportions. In other words, the 'diff' value represents $p_{top} - p_{bottom}$, where p_{top} is the proportion of total ink that is in the top half and p_{bottom} is the equivalent for the bottom half. A single pixel was defined to have ink if its value was greater than 200. We used this value because in the mnist_27 dataset they used the same benchmark. We took this approach because we noticed that 3s are generally more symmetrical over a 'x' axis, whereas 2s are less symmetrical. So, if more of the ink is in one half, it might be a 2, whereas if the ink of a digit is equally spread out among the top and bottom half we might expect it's a 3. We hypothesize that 3s will have a difference close to 0, while 2s will have a greater difference (probably heavier on the bottom).

```
#feature 2: horizontal symmetry of ink
#https://rdrr.io/cran/dslabs/src/inst/script/make-mnist_27.R

#Identifying indices have 2s or 3s
index_235 <- which(mnist$train$labels %in% c(2,3,5))
y <- mnist$train$labels[index_235]
x <- mnist$train$images[index_235,]

#Placeholder for splitting upper half and lower half
row_column <- expand_grid(row = 1:28, col = 1:28)
upper_half_idx <- which(row_column$row <= 14)
lower_half_idx <- which(row_column$row >14)

new_x <- x>200

new_x <- cbind(
  rowSums(new_x[,upper_half_idx]) / rowSums(new_x), # proportion of ink in top half
  rowSums(new_x[,lower_half_idx]) / rowSums(new_x) # proportion of ink in bottom half
)


f2.tbl <- as_tibble(new_x)%>%
  mutate(y = y,
         diff = V1 - V2, # feature 2
         abs_diff = abs(diff),
         index = index_235)%>%
  rename(top_prop = "V1",
         bottom_prop = "V2")%>%
    select(y, diff)
```

Dataset Creation

```
# combining first and second feature into a tibble
full.data235 <- tibble(
  y = f1.tbl$letter,
  x_1 = f1.tbl$feature.1,
  y_2 = f2.tbl$y,
  x_2 = f2.tbl$diff
)

set.seed(48494)
# filter out fives
full.data <- full.data235%>%
  filter(y %in% c("2","3"))%>%
  mutate(y = factor(y, levels = c("2","3")))%>%
  select(y, x_1,x_2)%>%
  slice_sample(n = 1000)

set.seed(48494)
# separating train and test data
split.data <- initial_split(full.data, prop = 0.8)
train.data <-training(split.data)
test.data  <-testing(split.data)
```

Modeling: 2's vs 3's

For models, we consider quadratic discriminate analysis and linear discriminate analysis.

Quadratic Discriminate Analysis

```
#qda
qda.model <- discrim_quad() %>%
  set_engine("MASS") %>%
  set_mode("classification")
```

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

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

qda.fit <- fit(qda.wflow, train.data)

pred.qda = qda.fit%>%
  augment(new_data = test.data)

# Accuracy for QDA
accuracy(pred.qda, test.data$y, pred.qda$.pred_class)[3]
```

```
## # A tibble: 1 x 1
##   .estimate
##       <dbl>
## 1     0.915
```

```
# Misclassification rate for QDA
1-as.numeric(accuracy(pred.qda, y,.pred_class)[3])
```

```
## [1] 0.085
```

```
# Confusion Matrix for QDA
conf_mat(pred.qda, y,.pred_class)
```

```
##           Truth
## Prediction  2  3
##          2 88 10
##          3  7 95
```

Linear Discriminate Analysis

```
#lda
lda.model <- discrim_linear() %>%
  set_engine("MASS") %>%
  set_mode("classification")

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

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

lda.fit <- fit(lda.wflow, train.data)

pred.lda = lda.fit%>%
  augment(new_data = test.data)

# Accuracy for LDA
accuracy(pred.lda, y,.pred_class)[3]
```

```
## # A tibble: 1 x 1
##   .estimate
##       <dbl>
## 1      0.91
```

```
# Misclassification rate for LDA
1-as.numeric(accuracy(pred.qda, y,.pred_class)[3])
```

```
## [1] 0.085
```

```
# Confusion matrix for LDA
conf_mat(pred.lda, y,.pred_class)
```

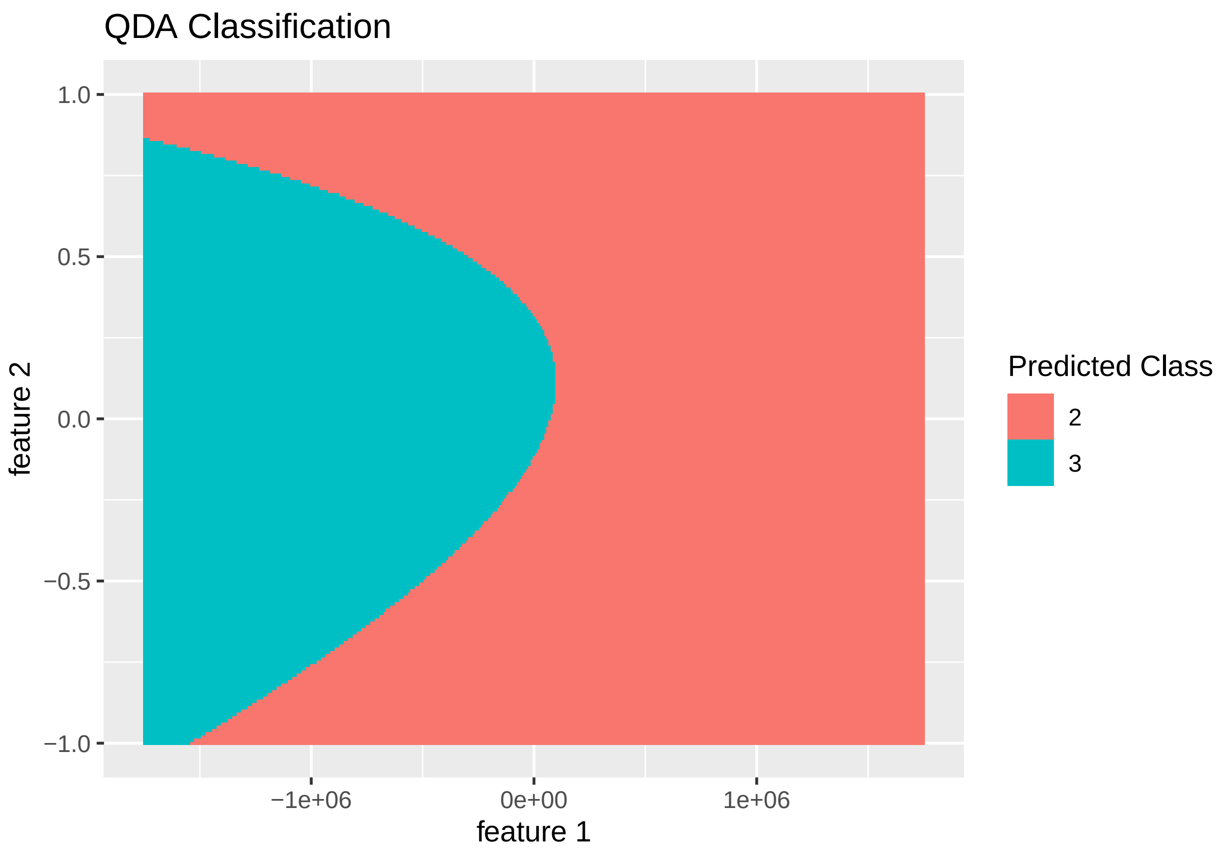
```
##           Truth
## Prediction  2  3
##          2 87 10
##          3  8 95
```

Both models do quite well, with accuracies above 90%. The QDA model has a slightly higher accuracy than LDA, so we select QDA are our preferred model.

Visualizations: 2's versus 3's

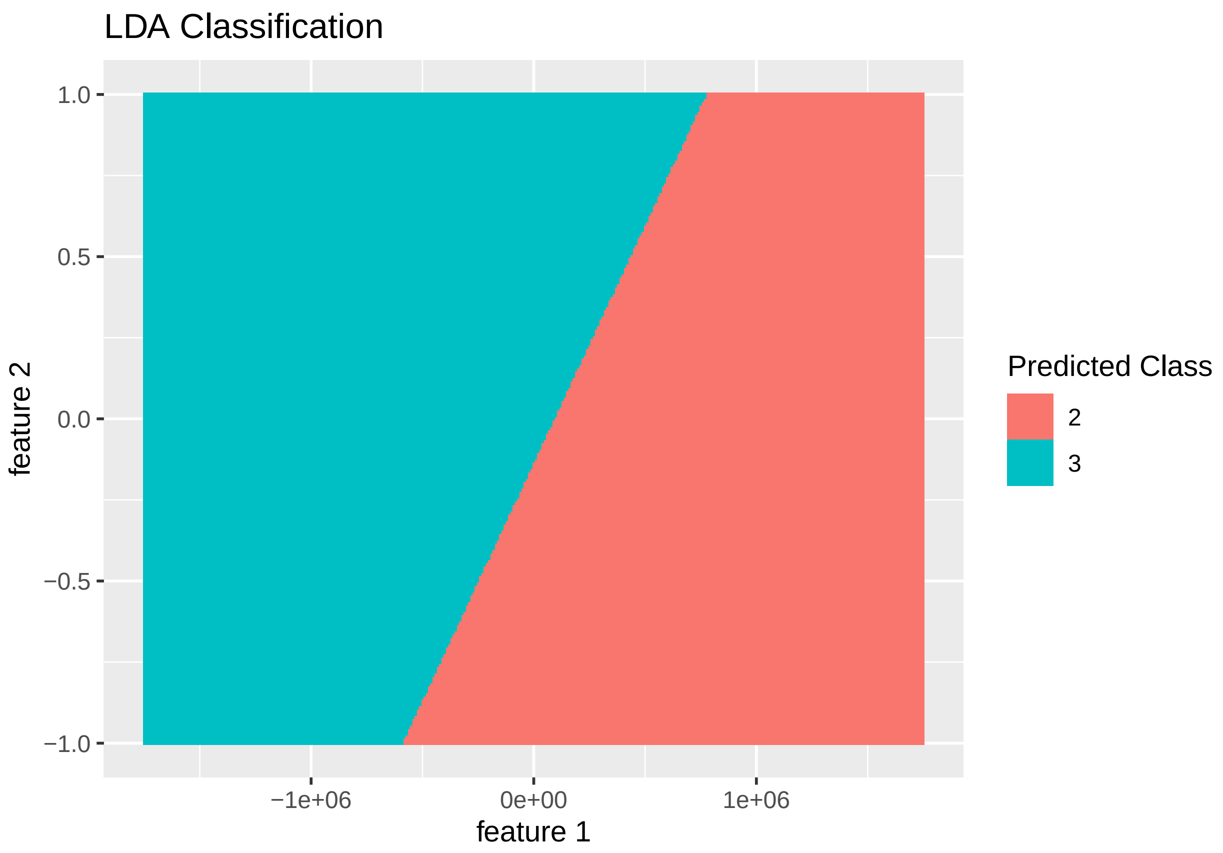
QDA

```
grid.vec.x = seq(-1750000,1750000, by=10000)
grid.vec.y = seq(-1,1, by=.01)

grid.tbl <- expand_grid(x_1=grid.vec.x, x_2=grid.vec.y)


pred.tbl = qda.fit%>%
  augment(grid.tbl)

pred.tbl%>%
  ggplot(aes(x = x_1,y=x_2,fill = .pred_class))+
  geom_raster()+
  labs(x = "feature 1", y = "feature 2", fill= "Predicted Class", title = "QDA Classification")
```



LDA

```
pred.tbl = lda.fit%>%
  augment(grid.tbl)

pred.tbl%>%
  ggplot(aes(x = x_1,y=x_2,fill = .pred_class))+
  geom_raster()+
  labs(x = "feature 1", y = "feature 2", fill= "Predicted Class", title = "LDA Classification")
```



Adding in 5’s

Data Creation

We calculated our features for the fives in the feature definition section to avoid repetative code, now we simply don’t filter them out.

```
set.seed(48494)

full.data <- full.data235%>%
  select(y, x_1,x_2)%>%
  slice_sample(n = 1000)

set.seed(48494)
split.data <- initial_split(full.data, prop = 0.8)
train.data <-training(split.data)
test.data <- testing(split.data)
```

Modeling: 2's, 3's, and 5's

Quadratic Discriminate Analysis

```
qda.model <- discrim_quad() %>%
  set_engine("MASS") %>%
  set_mode("classification")

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

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

qda.fit <- fit(qda.wflow, train.data)

pred.qda = qda.fit%>%
  augment(new_data = test.data)

# Accuracy for QDA with 5's
accuracy(pred.qda, y,.pred_class)[3]
```

```
## # A tibble: 1 x 1
##   .estimate
##       <dbl>
## 1      0.68
```

```
# Misclassification rate for QDA with 5's
1-as.numeric(accuracy(pred.qda, y,.pred_class)[3])
```

```
## [1] 0.32
```

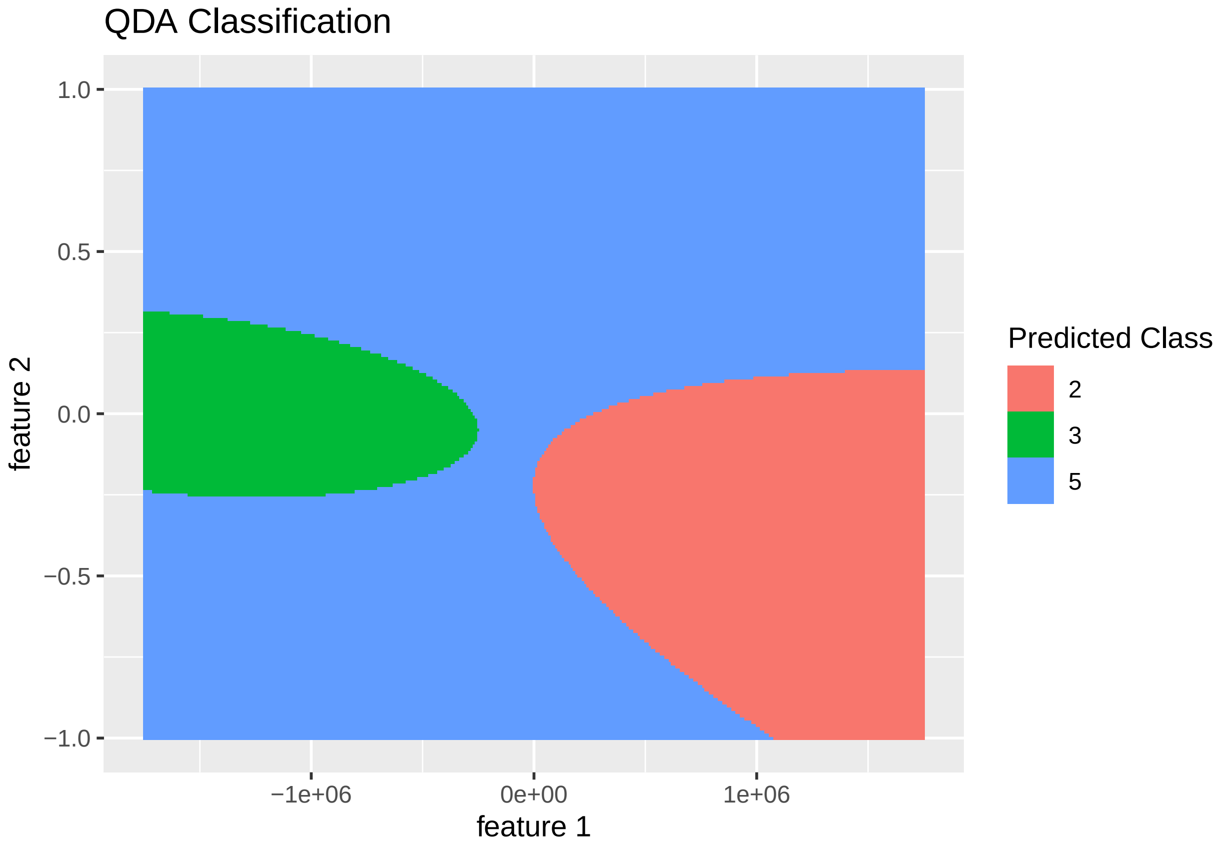
The performance of our model drops significantly when 5s are introduced to around 66%. Our first feature is defined in terms of the difference between 2s and 3s, so it can't isolate 5s in the same way. This is a fundamental limitation of the feature because differences can't be extended to more than two directions, although creating a new variable could address this. The second feature is defined in terms of symmetry of ink distribution, and 5s are most likely more symmetrical than 2s, so it makes sense that performance would be disrupted because of that as well.

```
# Confusion matrix for QDA with 5's
conf_mat(pred.qda, y,.pred_class)
```

```
##           Truth
## Prediction  2  3  5
##          2 62  0 12
##          3  0 57 35
##          5  6 11 17
```

The most common error is to mistake a 5 for a 3 followed by a 3 for a 5. This makes sense because 5's and 3's are more similar in terms of horizontal symmety and placement of dark pixels than 2's and 3's or 2's and 5's.

```
pred.tbl.qda5 = qda.fit%>%
  augment(grid.tbl)

pred.tbl.qda5%>%
  ggplot(aes(x = x_1,y=x_2,fill = .pred_class))+
  geom_raster()+
  labs(x = "feature 1", y = "feature 2", fill= "Predicted Class", title = "QDA Classification")
```



Linear Discriminate Analysis

```
#lda predicting 2,3,5
lda.model <- discrim_linear() %>%
  set_engine("MASS") %>%
  set_mode("classification")

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

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

lda.fit <- fit(lda.wflow, train.data)

pred.lda = lda.fit%>%
  augment(new_data = test.data)

# Accuracy for LDA with 5's
accuracy(pred.lda, y,.pred_class)[3]
```

```
## # A tibble: 1 x 1
##   .estimate
##       <dbl>
## 1     0.645
```

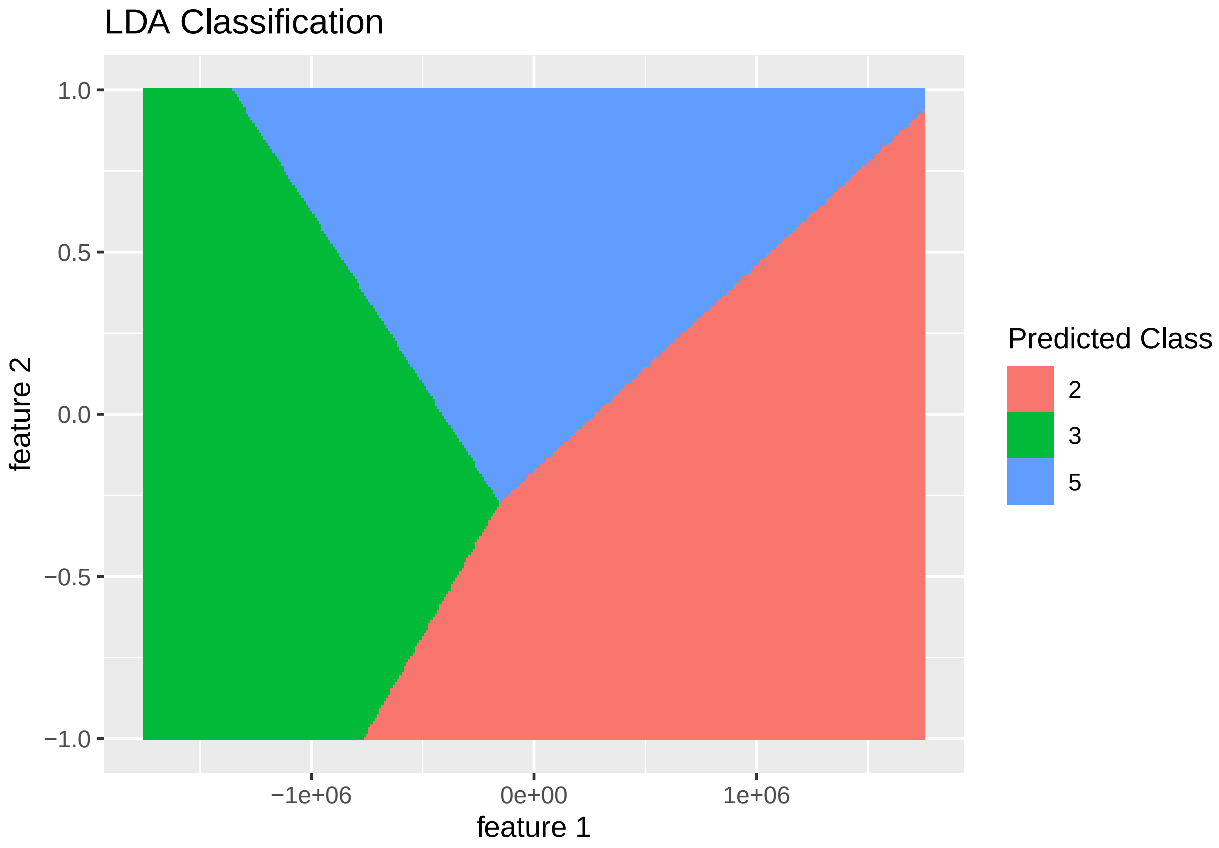
```
# Misclassification rate for LDA with 5's
1-as.numeric(accuracy(pred.qda, y,.pred_class)[3])
```

```
## [1] 0.32
```

```
# Confusion matrix for LDA with 5's
conf_mat(pred.lda, y,.pred_class)
```

```
##           Truth
## Prediction  2  3  5
##          2 60  0 11
##          3  1 56 40
##          5  7 12 13
```

```
pred.tbl.lda5 = lda.fit%>%
  augment(grid.tbl)

pred.tbl.lda5%>%
  ggplot(aes(x = x_1,y=x_2,fill = .pred_class))+
  geom_raster()+
  labs(x = "feature 1", y = "feature 2", fill= "Predicted Class", title = "LDA Classification")
```



LDA has similar performance to QDA with the addition of 5's.

Conclusion

We defined two features to classify between 2's and 3's in terms of differences in pixel averages and horizontal symmetry. Our two features are quite accurate at distingiushing between 2's and 3's, with an overall accuracy above 90% for both models. The features are less well suited at classification with the addition of 5's. In particular, our models often confuse 3's and 5's.