```
# -------------------------------------------------------------------------------
#
# Generative Models - case study - more than 3 classes
#
# -------------------------------------------------------------------------------

# Setup
library(tidyverse)
library(dslabs)
library(dplyr)
library(ggplot2)
library(Lahman)
library(HistData)
library(caret)
library(e1071)
library(matrixStats)

# In this video, we will give a slightly more complex example, one with three
# classes instead of two. We first create a dataset similar to the two or seven
# dataset. Except now we have one, twos, and sevens. We can generate that
# dataset using this rather complex code. Once we're done, we obtain training
# set and a test set.

if(!exists("mnist")) mnist <- read_mnist()

set.seed(3456)
index_127 <- sample(which(mnist$train$labels %in% c(1,2,7)), 2000)
y <- mnist$train$labels[index_127]
x <- mnist$train$images[index_127,]
index_train <- createDataPartition(y, p=0.8, list = FALSE)

# get the quandrants
# temporary object to help figure out the quandrants
row_column <- expand.grid(row=1:28, col=1:28)
upper_left_ind <- which(row_column$col <= 14 & row_column$row <= 14)
lower_right_ind <- which(row_column$col > 14 & row_column$row > 14)

# binarize the values. Above 200 is ink, below is no ink
x <- x > 200

# cbind proportion of pixels in upper right quandrant and proportion
# of pixes in lower rigth quandrant
x <- cbind(rowSums(x[ ,upper_left_ind])/rowSums(x),
           rowSums(x[ ,lower_right_ind])/rowSums(x))

train_set <- data.frame(y = factor(y[index_train]),
                        x_1 = x[index_train,1],
                        x_2 = x[index_train,2])
test_set <- data.frame(y = factor(y[-index_train]),
                       x_1 = x[-index_train,1],
                       x_2 = x[-index_train,2])

# Here we're showing the data for the training set.
train_set %>%
     ggplot(aes(x_1, x_2, color=y)) +
     geom_point()

# You can see the x1 and x2 predictors. And then in color, we're showing you the
# different labels, the different categories. The ones are in red, the greens
# are the twos, and the blue points are the sevens. As an example, we'll fit a
# qda model. We'll use the caret package. So all we do is type this piece of
# code.

train_qda <- train(y ~ .,
                   method = "qda",
                   data = train_set)

# So how do things differ now? First note that we estimate three conditional
# probabilities, although they all have to add up to 1.

# So if you type predict with type probability, you now get a matrix with three
# columns, a probability for the ones, a probability for the two, a probability
# for the sevens.

predict(train_qda, test_set, type = "prob") %>% head()
#            1         2          7
# 1 0.22232613 0.6596410 0.11803290
# 2 0.19256640 0.4535212 0.35391242
# 3 0.62749331 0.3220448 0.05046191
# 4 0.04623381 0.1008304 0.85293583
# 5 0.21671529 0.6229295 0.16035523
# 6 0.12669776 0.3349700 0.53833219

# We predict the one with the highest probability. So for the first observation,
# we would predict a two. And now our predictors are one of three classes.
```

```
predict(train_qda, test_set)
#>   [1] 2 2 1 7 2 7 2 1 7 1 7 7 2 7 7 1 1 7 1 7 7 1 7 7 7 7 1 1 1 2 1 7 2 7 1
#>  [36] 7 2 7 2 7 7 7 7 1 7 7 1 2 2 7 1 7 2 2 1 7 7 2 2 1 7 1 1 1 2 2 1 1 1 2
#>  [71] 7 1 7 7 2 1 7 1 7 2 7 1 2 2 1 2 2 2 1 1 1 7 1 2 1 7 2 2 2 7 1 1 1 7 1
#> [106] 7 7 7 2 7 1 7 7 2 1 2 1 1 2 1 2 2 1 2 1 7 1 1 2 1 7 2 7 1 2 1 2 1 2 1
#> [141] 1 1 7 2 2 7 1 1 2 2 2 7 7 1 7 1 7 7 2 1 2 7 7 1 7 7 7 1 7 7 2 2 1 2 2
#> [176] 1 2 2 2 1 2 7 2 7 7 7 1 7 7 2 2 1 2 1 2 7 1 7 7 1 1 1 7 1 1 7 2 1 7 7
#> [211] 2 2 1 7 2 2 2 2 7 2 2 1 2 2 1 1 1 7 1 7 1 7 7 7 2 7 1 1 2 2 1 7 1 7 7
#> [246] 7 1 1 7 1 1 7 2 1 2 1 2 1 7 1 7 1 2 7 7 7 7 7 2 1 7 1 7 2 1 7 7 1 7 7
#> [281] 7 2 7 1 2 7 2 2 7 2 2 7 2 1 2 1 1 1 7 1 1 7 7 1 7 2 1 7 7 7 7 1 2 2 7
#> [316] 7 1 1 7 1 2 1 2 1 7 7 1 1 1 7 1 2 7 7 1 1 7 2 7 7 7 1 7 7 7 7 7 2 2 1
#> [351] 7 7 2 1 2 2 7 7 1 7 7 1 1 7 7 1 1 1 2 1 7 2 7 2 7 1 2 2 1 1 7 2 7 2 1
#> [386] 2 7 7 7 2 7 1 1 7 1 7 2 7 7
#> Levels: 1 2 7

# If we use the predict function, with the default setting of just giving you
# the outcome, we get twos, ones, and sevens. The confusion matrix is a
# three-by-three table now because we can make two kinds of mistakes with the
# ones, two kinds of mistakes with the two, and two kinds of mistakes with the
# sevens. You can see it here.

confusionMatrix(predict(train_qda, test_set), test_set$y)
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction   1   2   7
#>          1 111  17   7
#>          2  14  80  17
#>          7  19  25 109
#> 
#> Overall Statistics
#> 
#>                Accuracy : 0.752
#>                  95% CI : (0.706, 0.794)
#>     No Information Rate : 0.361
#>     P-Value [Acc > NIR] : <2e-16
#> 
#>                   Kappa : 0.627
#>  Mcnemar's Test P-Value : 0.0615
#> 
#> Statistics by Class:
#> 
#>                      Class: 1 Class: 2 Class: 7
#> Sensitivity             0.771    0.656    0.820
#> Specificity             0.906    0.888    0.835
#> Pos Pred Value          0.822    0.721    0.712
#> Neg Pred Value          0.875    0.854    0.902
#> Prevalence              0.361    0.306    0.333
#> Detection Rate          0.278    0.201    0.273
#> Detection Prevalence    0.338    0.278    0.383
#> Balanced Accuracy       0.838    0.772    0.827

# The accuracy's still at one number because it just basically computes how
# often we make the correct prediction.

confusionMatrix(predict(train_qda, test_set), test_set$y)$overal["Accuracy"]
#> Accuracy 
#>    0.752

# Note that for sensitivity and specificity, we have a pair of values for each
# class. This is because to define these terms, we need a binary outcome. We
# therefore have three columns, one for each class as a positive and the other
# two as the negatives. Finally, we can visualize what parts of the regions are
# called ones, twos, and seven by simply plotting the estimated conditional
# probability.

GS <- 150
new_x <- expand.grid(x_1 = seq(min(train_set$x_1), max(train_set$x_1), len=GS),
                     x_2 = seq(min(train_set$x_2), max(train_set$x_2), len=GS))
new_x %>% mutate(y_hat = predict(train_qda, new_x)) %>%
     ggplot(aes(x_1, x_2, color = y_hat, z = as.numeric(y_hat))) +
     geom_point(size = 0.5, pch = 16) +
     stat_contour(breaks=c(1.5, 2.5),color="black") +
     guides(colour = guide_legend(override.aes = list(size=2)))

# Let's see how it looks like for lda. # We can train the model like this.

train_lda <- train(y ~ .,
                   method = "lda",
                   data = train_set)

confusionMatrix(predict(train_lda, test_set), test_set$y)$overal["Accuracy"]
#> Accuracy 
#>    0.664
```

```
# The accuracy is much worse, and it is because our boundary regions have three
# lines. This is something we can show mathematically. The results for knn are
# actually much better.

# INSERT CODE FOR GRAPH

# The results for kNN are much better:

train_knn <- train(y ~ .,
                   method = "knn",
                   tuneGrid = data.frame(k = seq(15, 51, 2)),
                   data = train_set)

confusionMatrix(predict(train_knn, test_set), test_set$y)$overal["Accuracy"]
#> Accuracy
#>    0.769

# Look how higher the accuracy is.

# And we can also see that the estimated conditional probability is much more
# flexible, as we can see in this plot.

new_x %>% mutate(y_hat = predict(train_knn, new_x)) %>%
     ggplot(aes(x_1, x_2, color = y_hat, z = as.numeric(y_hat))) +
     geom_point(size = 0.5, pch = 16) +
     stat_contour(breaks=c(1.5, 2.5),color="black") +
     guides(colour = guide_legend(override.aes = list(size=2)))

# Note that the reason that qda and, in particularly, lda are not working well
# is due to lack of fit. We can see that by plotting the data and noting that at
# least the ones are definitely not bivariate normally distributed.

train_set %>% mutate(y = factor(y)) %>%
     ggplot(aes(x_1, x_2, fill = y, color=y)) +
     geom_point(show.legend = FALSE) +
     stat_ellipse(type="norm")

# So in summary, generating models can be very powerful but only when we're able
# to successfully approximate the joint distribution of predictor's condition on
# each class.
```