lab04

Sean Fitch

2024-10-04

```r
# Load the Wine dataset
data(wine)
```

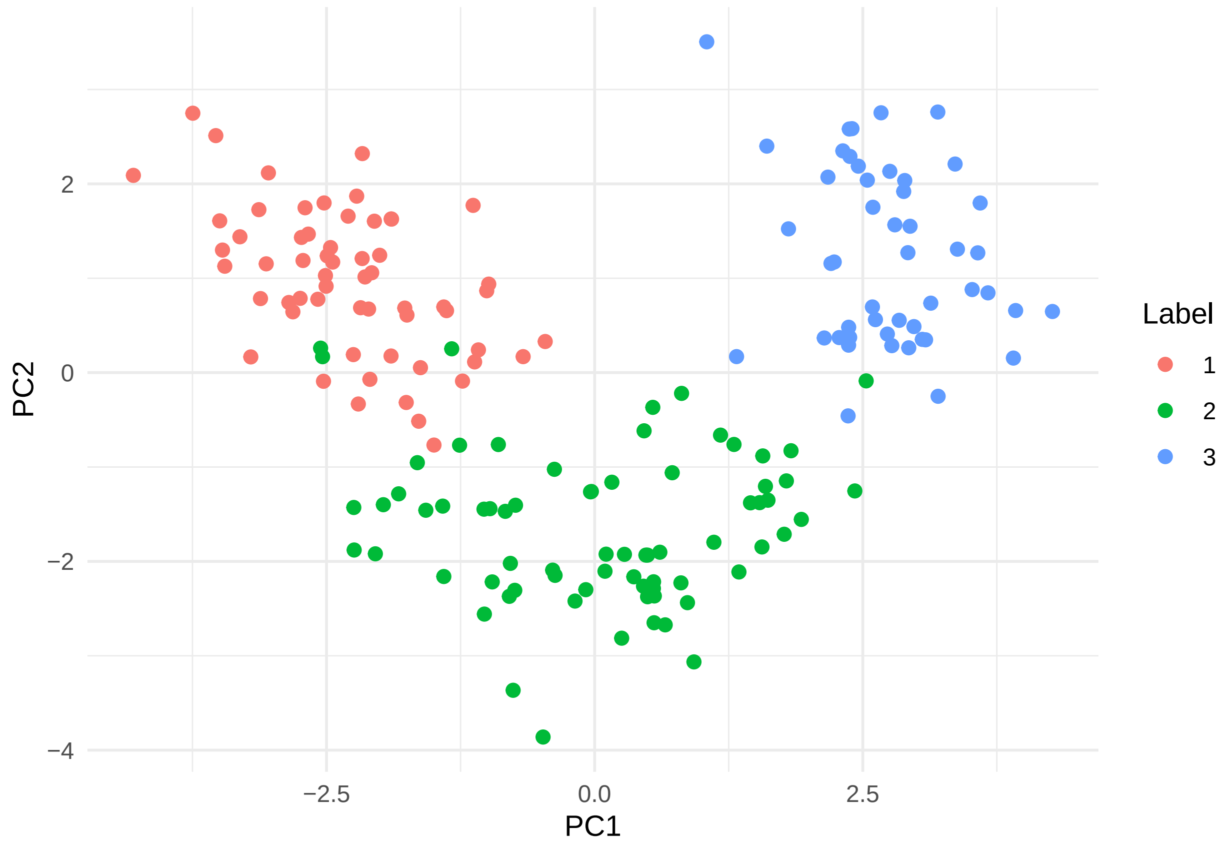
```r
pca_result <- prcomp(subset(wine, select = -Type), scale. = TRUE)
pca_data <- as.data.frame(pca_result$x)

# plot_pca_3d <- function(pca_data, colors, title) {
#   p <- plot_ly()
#
#   # Add rays from the origin (0,0,0) to each point
#   for (i in 1:nrow(pca_data)) {
#     p <- p %>%
#       add_trace(x = c(0, pca_data$PC1[i]),
#                 y = c(0, pca_data$PC2[i]),
#                 z = c(0, pca_data$PC3[i]),
#                 type = "scatter3d", mode = "lines",
#                 line = list(color = 'gray', width = 1),
#                 showlegend = FALSE)
#   }
#
#   # Add the points
#   p <- p %>%
#     add_trace(data = pca_data,
#               x = ~PC1, y = ~PC2, z = ~PC3,
#               color = colors,
#               colors = "Set1",
#               type = "scatter3d", mode = "markers") %>%
#     layout(title = title,
#            scene = list(xaxis = list(title = 'PC1'),
#                         yaxis = list(title = 'PC2'),
#                         zaxis = list(title = 'PC3')))
#
#   return(p)
# }

plot_pca_2d <- function(data, labels) {
  # Ensure the labels data frame has the same number of rows as data
  if (nrow(data) != length(labels)) {
    stop("Labels must have the same number of rows as the data.")
  }

  # Plot
  library(ggplot2)
  ggplot(data.frame(x = data[[1]], y = data[[2]], label = labels), aes(x = x, y = y, color = label)) +
    geom_point(size = 2) +
    labs(x = "PC1", y = "PC2", color = "Label") +
    theme_minimal()
}

plot_pca_2d(as.data.frame(pca_result$x), wine$Type)
```



```
# plot_pca_3d(as.data.frame(pca_result$x[, 1:3]), wine$Type, "3D PCA")
```

```
pc1_loadings <- pca_result$rotation[, 1]
sort(abs(pc1_loadings), decreasing = TRUE)
```

```
##      Flavanoids         Phenols        Dilution Proanthocyanins    Nonflavanoids
##     0.422934297     0.394660845     0.376167411     0.313429488     0.298533103
##             Hue         Proline           Malic      Alcalinity         Alcohol
##     0.296714564     0.286752227     0.245187580     0.239320405     0.144329395
##       Magnesium           Color             Ash
##     0.141992042     0.088616705     0.002051061
```

The largest contributor is Flavanoids followed by Phenols

```r
train_and_evaluate_nn <- function(features, labels, learning_rate = 0.001) {
  # Normalize the features
  features_normalized <- scale(features)
  features <- as.matrix(features_normalized)
  labels <- as.integer(labels)

  # Convert to torch tensors
  x_tensor <- torch_tensor(features, dtype = torch_float())
  y_tensor <- torch_tensor(labels, dtype = torch_long())

  # Split the dataset with stratification
  set.seed(42)
  n_samples <- nrow(features)
  n_classes <- length(unique(labels))

  # Stratified sampling
  train_index <- c()
  for (class in 1:n_classes) {
    class_indices <- which(labels == class)
    n_class_samples <- length(class_indices)
    n_train_samples <- floor(0.8 * n_class_samples)
    train_index <- c(train_index,
                     sample(class_indices, size = n_train_samples))
  }
  test_index <- setdiff(1:n_samples, train_index)

  # Create train and test sets
  x_train <- x_tensor[train_index, ]
  y_train <- y_tensor[train_index]
  x_test <- x_tensor[test_index, ]
  y_test <- y_tensor[test_index]

  model <- nn_module(
    initialize = function() {
      self$fc1 <- nn_linear(ncol(features), 16)
      self$dropout1 <- nn_dropout(0.3)
      self$fc2 <- nn_linear(16, 16)
      self$dropout2 <- nn_dropout(0.3)
      self$fc3 <- nn_linear(16, 8)
      self$fc4 <- nn_linear(8, n_classes)
      self$batch_norm1 <- nn_batch_norm1d(16)
      self$batch_norm2 <- nn_batch_norm1d(16)
    },

    forward = function(x) {
      x %>%
        self$fc1() %>%
        self$batch_norm1() %>%
        nnf_relu() %>%
        self$dropout1() %>%
        self$fc2() %>%
        self$batch_norm2() %>%
        nnf_relu() %>%
        self$dropout2() %>%
        self$fc3() %>%
        nnf_relu() %>%
        self$fc4() %>%
        nnf_log_softmax(dim = 1)
    }
  )

  # Instantiate the model
  model <- model()

  # Define loss function and optimizer
  criterion <- nn_cross_entropy_loss()
  optimizer <- optim_adam(model$parameters, lr = learning_rate)

  # Training loop with early stopping
  best_loss <- Inf
  patience_counter <- 0
  max_patience <- 10
  min_lr <- 1e-6

  for (epoch in 1:100) {
    model$train()
    optimizer$zero_grad()

    # Forward pass
    output <- model(x_train)
    loss <- criterion(output, y_train)

    # Backward pass and optimization
    loss$backward()
    optimizer$step()

    # Manual learning rate adjustment if loss plateaus
    if (loss$item() >= best_loss) {
      patience_counter <- patience_counter + 1
      if (patience_counter >= 5) {
        current_lr <- optimizer$param_groups[[1]]$lr
        if (current_lr > min_lr) {
          new_lr <- current_lr * 0.5
          optimizer$param_groups[[1]]$lr <- new_lr
          cat(sprintf("Reducing learning rate to %f\n", new_lr))
        }
        patience_counter <- 0
      }
    } else {
      best_loss <- loss$item()
      patience_counter <- 0
    }

    # Early stopping check
    if (optimizer$param_groups[[1]]$lr <= min_lr && patience_counter >= max_patience) {
      cat(sprintf("Early stopping at epoch %d\n", epoch))
      break
    }

    # Print progress
    if (epoch %% 10 == 0) {
      cat(sprintf("Epoch [%d/100], Loss: %.4f, LR: %.6f\n",
                 epoch, loss$item(), optimizer$param_groups[[1]]$lr))
    }
  }

  # Evaluate the model
  model$eval()
  with_no_grad({
    output <- model(x_test)
    predicted_probs <- output$exp()
    max_result <- predicted_probs$max(dim = 2)
    predicted_classes <- max_result[[2]]

    correct <- sum(predicted_classes == y_test)
    accuracy <- correct$item() / length(y_test)

    # Create confusion matrix
    conf_matrix <- matrix(0, nrow = n_classes, ncol = n_classes)
    pred_vec <- as.array(predicted_classes$cpu())
    true_vec <- as.array(y_test$cpu())

    for (i in 1:length(pred_vec)) {
      conf_matrix[true_vec[i], pred_vec[i]] <- conf_matrix[true_vec[i], pred_vec[i]] + 1
    }

    rownames(conf_matrix) <- paste("Actual", 1:n_classes)
    colnames(conf_matrix) <- paste("Predicted", 1:n_classes)
  })

  # Print results
  cat('\nTest accuracy:', accuracy, "\n")
  cat('\nConfusion Matrix:\n')
  print(conf_matrix)

  # Calculate and print additional metrics
  cat('\nPer-class metrics:\n')
  for (i in 1:n_classes) {
    tp <- conf_matrix[i,i]
    fp <- sum(conf_matrix[,i]) - tp
    fn <- sum(conf_matrix[i,]) - tp

    precision <- tp / (tp + fp)
    recall <- tp / (tp + fn)
    f1 <- 2 * (precision * recall) / (precision + recall)

    cat(sprintf("\nClass %d:\n", i))
    cat(sprintf("Precision: %.3f\n", precision))
    cat(sprintf("Recall: %.3f\n", recall))
```

```
    cat(sprintf("F1 Score: %.3f\n", f1))
  }
}
```

```
train_and_evaluate_nn(subset(wine, select = -Type), wine$Type)
```

```
## Epoch [10/100], Loss: 1.0706, LR: 0.001000
## Epoch [20/100], Loss: 1.0348, LR: 0.001000
## Epoch [30/100], Loss: 0.9843, LR: 0.001000
## Epoch [40/100], Loss: 0.9096, LR: 0.001000
## Epoch [50/100], Loss: 0.8611, LR: 0.001000
## Epoch [60/100], Loss: 0.8100, LR: 0.001000
## Epoch [70/100], Loss: 0.7463, LR: 0.001000
## Epoch [80/100], Loss: 0.6446, LR: 0.001000
## Epoch [90/100], Loss: 0.5889, LR: 0.001000
## Epoch [100/100], Loss: 0.5039, LR: 0.001000
##
## Test accuracy: 0.972973
##
## Confusion Matrix:
##          Predicted 1 Predicted 2 Predicted 3
## Actual 1          12           0           0
## Actual 2           1          14           0
## Actual 3           0           0          10
##
## Per-class metrics:
##
## Class 1:
## Precision: 0.923
## Recall: 1.000
## F1 Score: 0.960
##
## Class 2:
## Precision: 1.000
## Recall: 0.933
## F1 Score: 0.966
##
## Class 3:
## Precision: 1.000
## Recall: 1.000
## F1 Score: 1.000
```

```
train_and_evaluate_nn(pca_data[1:3], wine$Type)
```

```
## Epoch [10/100], Loss: 1.0601, LR: 0.001000
## Epoch [20/100], Loss: 1.0316, LR: 0.001000
## Epoch [30/100], Loss: 1.0000, LR: 0.001000
## Epoch [40/100], Loss: 0.9419, LR: 0.001000
## Epoch [50/100], Loss: 0.9187, LR: 0.001000
## Epoch [60/100], Loss: 0.8257, LR: 0.001000
## Reducing learning rate to 0.000500
## Epoch [70/100], Loss: 0.8041, LR: 0.000500
## Reducing learning rate to 0.000250
## Reducing learning rate to 0.000125
## Epoch [80/100], Loss: 0.7743, LR: 0.000125
## Epoch [90/100], Loss: 0.7461, LR: 0.000125
## Reducing learning rate to 0.000063
## Epoch [100/100], Loss: 0.7711, LR: 0.000063
##
## Test accuracy: 0.8918919
##
## Confusion Matrix:
##            Predicted 1 Predicted 2 Predicted 3
## Actual 1            11           0           1
## Actual 2             0          12           3
## Actual 3             0           0          10
##
## Per-class metrics:
##
## Class 1:
## Precision: 1.000
## Recall: 0.917
## F1 Score: 0.957
##
## Class 2:
## Precision: 1.000
## Recall: 0.800
## F1 Score: 0.889
##
## Class 3:
## Precision: 0.714
## Recall: 1.000
## F1 Score: 0.833
```

```
wine_top <- wine %>% select(-Ash, -Color, -Magnesium, -Alcohol)
pca_result_top <- prcomp(subset(wine_top, select = -Type), scale. = TRUE)
pca_data_top <- as.data.frame(pca_result_top$x)
```

```
train_and_evaluate_nn(pca_data_top[1:3], wine$Type)
```

```
## Epoch [10/100], Loss: 1.0998, LR: 0.001000
## Epoch [20/100], Loss: 1.0700, LR: 0.001000
## Epoch [30/100], Loss: 1.0556, LR: 0.001000
## Epoch [40/100], Loss: 1.0202, LR: 0.001000
## Epoch [50/100], Loss: 0.9722, LR: 0.001000
## Epoch [60/100], Loss: 0.9275, LR: 0.001000
## Epoch [70/100], Loss: 0.8858, LR: 0.001000
## Epoch [80/100], Loss: 0.8058, LR: 0.001000
## Epoch [90/100], Loss: 0.7923, LR: 0.001000
## Reducing learning rate to 0.000500
## Epoch [100/100], Loss: 0.7084, LR: 0.000500
##
## Test accuracy: 0.9189189
##
## Confusion Matrix:
```

```
##          Predicted 1 Predicted 2 Predicted 3
## Actual 1          11           1           0
## Actual 2           0          14           1
## Actual 3           0           1           9
##
## Per-class metrics:
##
## Class 1:
## Precision: 1.000
## Recall: 0.917
## F1 Score: 0.957
##
## Class 2:
## Precision: 0.875
## Recall: 0.933
## F1 Score: 0.903
##
## Class 3:
## Precision: 0.900
## Recall: 0.900
## F1 Score: 0.900
```