607 Week 2B Assignment

Pricilla

2025-09-04

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.0     ✓ stringr   1.5.1
## ✓ ggplot2   3.5.2     ✓ tibble    3.3.0
## ✓ lubridate 1.9.4     ✓ tidyr     1.3.1
## ✓ purrr     1.1.0
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✗ dplyr::filter() masks stats::filter()
## ✗ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
e errors
```

```
library(dplyr)
library(readr)
library(ggplot2)
library(tibble)
```

##1.1 Load the penguin data into a data frame and preview.

```
peng_data <- read_csv("https://raw.githubusercontent.com/prnakyazze94/Data_607/refs/heads/main/p
enguin_predictions.csv.csv")
```

```
## Rows: 93 Columns: 3
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (2): .pred_class, sex
## dbl (1): .pred_female
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
glimpse(peng_data)
```

```
## Rows: 93
## Columns: 3
## $ .pred_female <dbl> 0.99217462, 0.95423945, 0.98473504, 0.18702056, 0.9947012…
## $ .pred_class  <chr> "female", "female", "female", "male", "female", "female",…
## $ sex          <chr> "female", "female", "female", "female", "female", "female…
```

```
table(peng_data$sex)
```

```
##
## female   male
##     39     54
```

Actual Female = 39

Actual Male = 54

Total = 93

Determine the majority class

Majority class = Male (54 occurrences)

So if we always predicted “Male”, we’d be correct 54 times

We’d be wrong on all the Females → 39 errors

Calculate the Null Error Rate

39 divide by 93 = 0.419

If you did nothing but always guess “Male”, you’d be wrong about 41.19% of the time.

A model must beat this error rate to be considered useful.

Accuracy of model

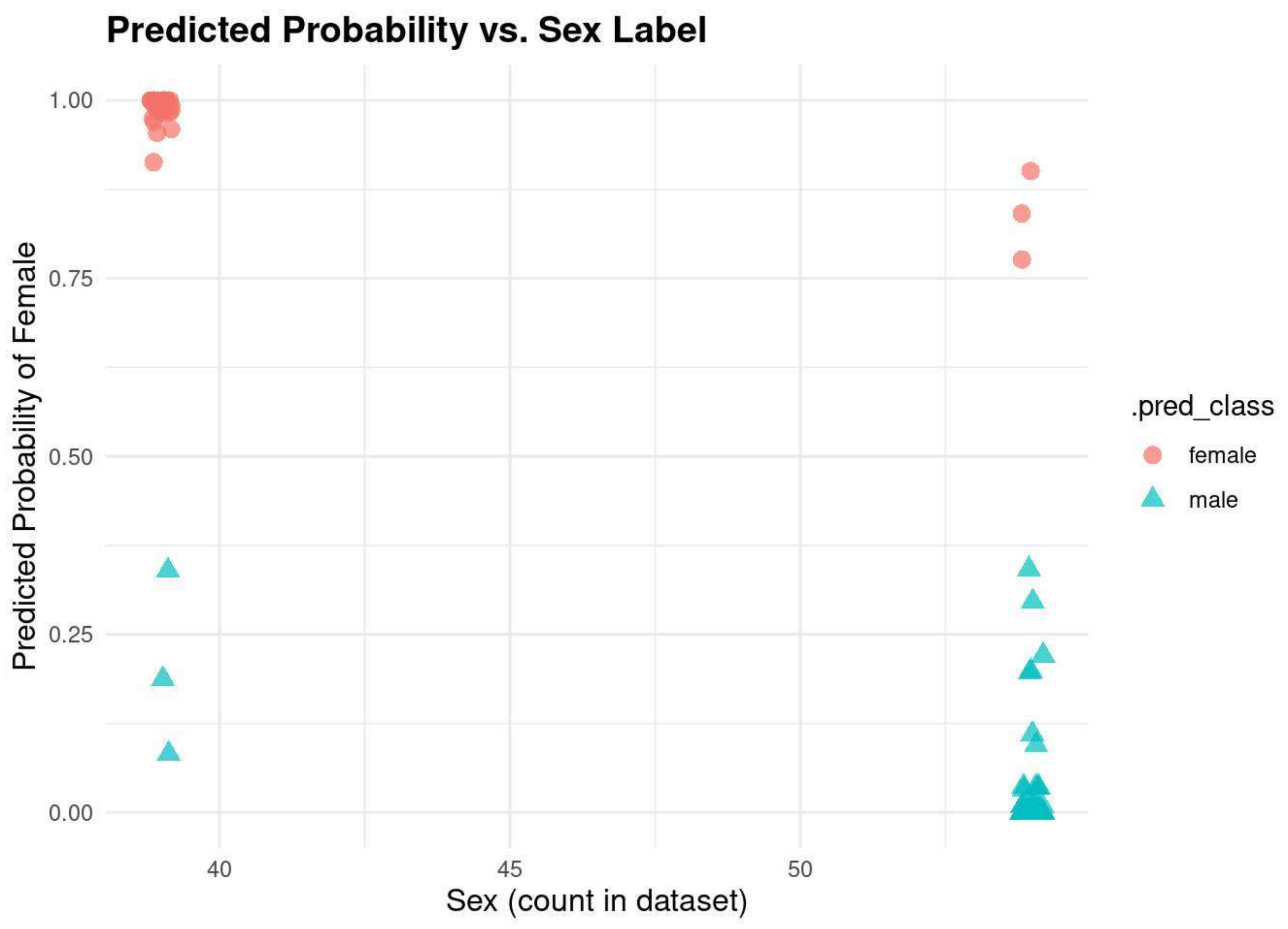
The model predicted 93 out of 100 correctly.

```
accuracy <- mean(peng_data$.pred_class == peng_data$sex)

cat("accuracy:", round(accuracy, 4), "\n")
```

```
## accuracy: 0.9355
```

Predicted Probability vs. Sex Label PLOT

```
# Create a proxy, how many times each sex #label appears
peng <- peng_data %>%
  count(sex, name = "peng")

# Merge with peng_data
peng_plot_df <- peng_data %>%
  left_join(peng, by = "sex") %>%
  mutate(Model = "Penguin Classifier")

# Plot predicted probability vs. RealSex
ggplot(peng_plot_df, aes(x = peng, y = .pred_female, color = .pred_class, shape = .pred_class))
+
  geom_jitter(width = 0.2, height = 0, size = 3, alpha = 0.7) +
  labs(
    title = "Predicted Probability vs. Sex Label",
    x = "Sex (count in dataset)",
    y = "Predicted Probability of Female"
  ) +
  theme_minimal() +
  theme(
    legend.position = "right",
    plot.title = element_text(size = 14, face = "bold"),
    axis.title = element_text(size = 12)
  )
```



EXPLANATION

Predicted Class vs Sex

Female Predictions.

Predicted probabilities are tightly clustered near 1.0.

High accurate predictions are dominated by females in the dataset.

There is very little spread or uncertainty among female predictions meaning the model is very confident when it predicts “female”.

Male Predictions.

Predicted probabilities are clustered near 0.0–0.3.

The model predicts “male” in this group, but with slightly more variation than in the female class.

A wider spread exists for male predictions, but still clearly below the 0.5 decision threshold.

```
#Add binary column as the 4th column
peng_data <- peng_data %>%
  mutate(actual_binary = ifelse(sex == "female", 1, 0)) %>%
  relocate(actual_binary, .after = 3)  # Places it as the 4th column
glimpse(peng_data)
```

```
## Rows: 93
## Columns: 4
## $ .pred_female  <dbl> 0.99217462, 0.95423945, 0.98473504, 0.18702056, 0.994701…
## $ .pred_class   <chr> "female", "female", "female", "male", "female", "female"…
## $ sex           <chr> "female", "female", "female", "female", "female", "femal…
## $ actual_binary <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,…
```

1.Calculate and state the null error rate for the provided classification_model_performance.csv dataset.

```
#  Count each class in the sex column
class_counts <- table(peng_data$sex)

# Get the number of majority class examples (max count)
majority_class_count <- max(class_counts)

# Calculate total number of records
total <- sum(class_counts)

# Calculate number of errors (i.e., non-majority cases)
errors <- total - majority_class_count

# Calculate null error rate
null_error_rate <- errors / total

# Print result
cat("Null Error Rate:", round(null_error_rate, 4), "\n")
```

```
## Null Error Rate: 0.4194
```

NULL error using binary

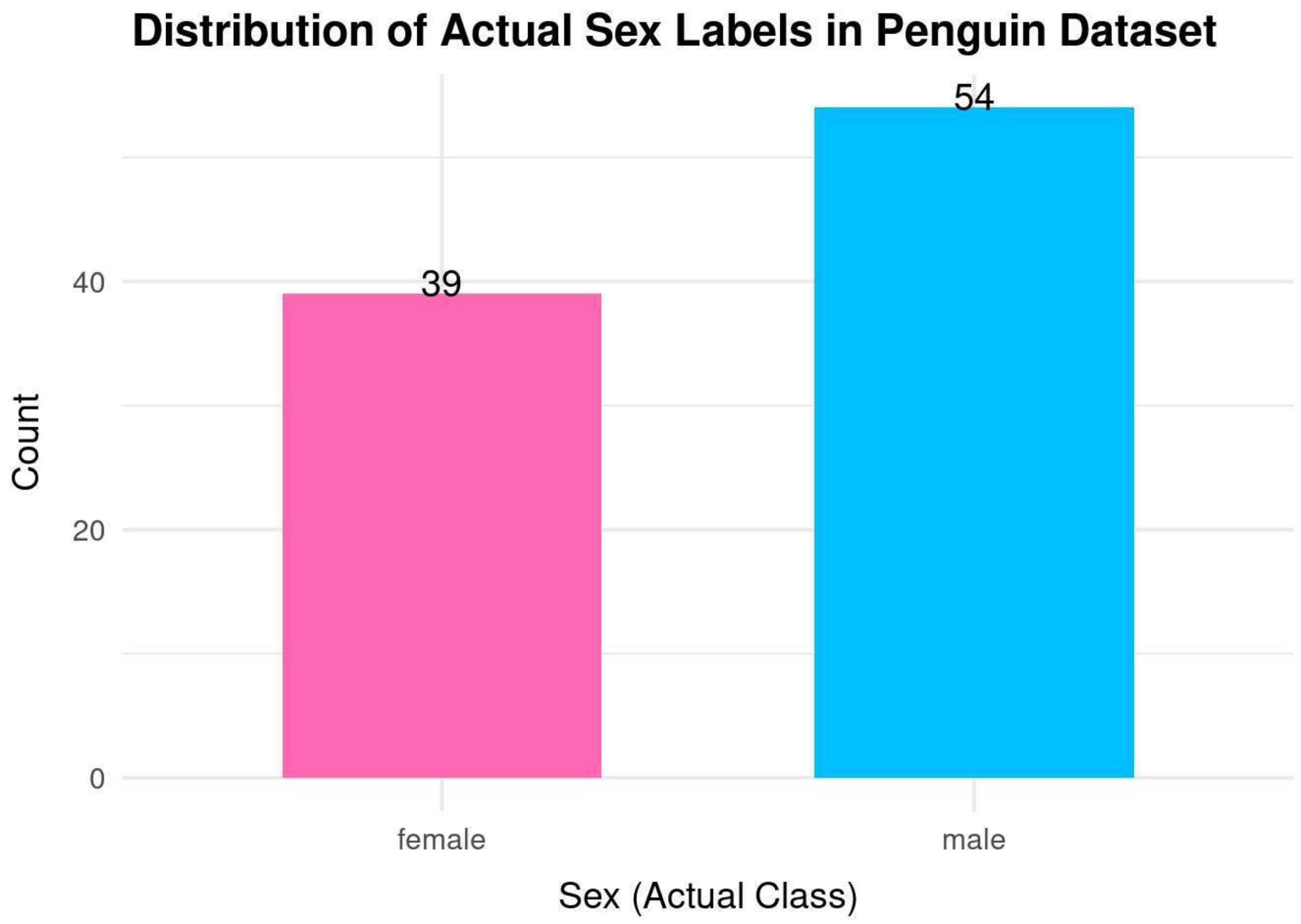
```
# Convert actual gender to binary: female = 1, male = 0
peng_data <- peng_data %>%
  mutate(
    actual_binary = ifelse(sex == "female", 1, 0),
    predicted_binary = ifelse(.pred_class == "female", 1, 0)
  )

# Compute null error rate
majority_class <- peng_data %>%
  count(actual_binary) %>%
  arrange(desc(n)) %>%
  slice(1)

majority_class_prop <- majority_class$n / nrow(peng_data)
null_error_rate <- 1 - majority_class_prop

cat(round(null_error_rate, 4), "\n")
```

```
## 0.4194
```

It is important to know the Null error rate is the error you’d make by always guessing the majority, without using any model. If a model’s accuracy is worse than the null error rate, it’s worse than guessing. If one class dominates, a model can be highly accurate but totally biased, just by always predicting the majority class.

Create a plot showing the data distribution of the actual explanatory variable.

```
# Optional: Clean label names and count
peng_data %>%
  count(sex) %>%
  ggplot(aes(x = sex, y = n, fill = sex)) +
  geom_col(width = 0.6, show.legend = FALSE) +
  geom_text(aes(label = n), vjust = 0.1, size = 5) +
  scale_fill_manual(values = c("female" = "#FF69B4", "male" = "#00BFFF")) +
  labs(
    title = "Distribution of Actual Sex Labels in Penguin Dataset",
    x = "Sex (Actual Class)",
    y = "Count"
  ) +
  theme_minimal(base_size = 14) +
  theme(
    plot.title = element_text(face = "bold", hjust = 0.1),
    axis.title.x = element_text(margin = margin(t = 10)),
    axis.title.y = element_text(margin = margin(r = 10))
  )
```



Using yardstick for more metrics. Using the binaryestimator the estimated accuracy is 93% out of 100..

```
library(yardstick)
```

```
##
## Attaching package: 'yardstick'
```

```
## The following object is masked from 'package:readr':
##
##     spec
```

```
# Ensure columns are factors
peng_data <- peng_data %>%
  mutate(
    sex = factor(sex),
    .pred_class = factor(.pred_class)
  )

# Compute accuracy
accuracy_result <- accuracy(peng_data, truth = sex, estimate = .pred_class)
print(accuracy_result)
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.935
```

Labeled Confusion Matrix

```
# Generate confusion matrix
conf_tbl <- table(Actual = peng_data$sex, Predicted = peng_data$.pred_class)

# Extract counts
TP <- conf_tbl["female", "female"]
FN <- conf_tbl["female", "male"]
FP <- conf_tbl["male", "female"]
TN <- conf_tbl["male", "male"]

# Create a labeled matrix
labeled_matrix <- matrix(
  c(paste0("TP = ", TP), paste0("FN = ", FN),
    paste0("FP = ", FP), paste0("TN = ", TN)),
  nrow = 2, byrow = TRUE,
  dimnames = list(
    c("Actual Female", "Actual Male"),
    c("Predicted Female", "Predicted Male")
  )
)

# Print labeled confusion matrix
print(labeled_matrix)
```

```
##               Predicted Female Predicted Male
## Actual Female "TP = 36"        "FN = 3"
## Actual Male   "FP = 3"         "TN = 51"
```

2. Analyze the data to determine the true positive, false positive, true negative, and false negative values for the dataset, using .pred_female thresholds of 0.2, 0.5, and 0.8. Display your results in three confusion matrices, with the counts of TP, FP, TN, and FN.

The confusion matrix terminology means.

TP (True Positive) is Actual Female predicted as Female. FN (False Negative) is Actual Female predicted as Male (missed a Female). FP (False Positive) is Actual Male predicted as Female (mistakenly called Female). TN (True Negative) is Actual Male predicted as Male.

Predicting Female means the model thinks the person is Female.

Predicting Male means the model thinks the person is Male.

```
# Create a function to compute confusion matrix at a given threshold
get_confusion_matrix <- function(data, threshold) {
  data <- data %>%
    mutate(
      predicted_binary = ifelse(.pred_female >= threshold, 1, 0),  # Predict female if prob ≥ th
reshold
      actual_binary = ifelse(sex == "female", 1, 0)                # 1 = female, 0 = male
    )

  # Calculate confusion matrix values
  TP <- sum(data$predicted_binary == 1 & data$actual_binary == 1)
  FP <- sum(data$predicted_binary == 1 & data$actual_binary == 0)
  TN <- sum(data$predicted_binary == 0 & data$actual_binary == 0)
  FN <- sum(data$predicted_binary == 0 & data$actual_binary == 1)

  # Create labeled matrix
  matrix(
    c(paste0("TP = ", TP), paste0("FN = ", FN),
      paste0("FP = ", FP), paste0("TN = ", TN)),
    nrow = 2, byrow = TRUE,
    dimnames = list(
      c("Actual Female", "Actual Male"),
      c("Predicted Female", "Predicted Male")
    )
  )
}

# Generate and print confusion matrices at different thresholds
thresholds <- c(0.2, 0.5, 0.8)

for (t in thresholds) {
  cat("\n Confusion Matrix at Threshold:", t, "\n")
  print(get_confusion_matrix(peng_data, t))
}
```

```
##
##  Confusion Matrix at Threshold: 0.2
##               Predicted Female Predicted Male
## Actual Female "TP = 37"        "FN = 2"
## Actual Male   "FP = 6"         "TN = 48"
##
##  Confusion Matrix at Threshold: 0.5
##               Predicted Female Predicted Male
## Actual Female "TP = 36"        "FN = 3"
## Actual Male   "FP = 3"         "TN = 51"
##
##  Confusion Matrix at Threshold: 0.8
##               Predicted Female Predicted Male
## Actual Female "TP = 36"        "FN = 3"
## Actual Male   "FP = 2"         "TN = 52"
```

Threshold 0.2.or 20%

The model is very lenient for it only needs 20% confidence to say “Female”.

More TPs (detects more females): 37

But also more FPs (wrongly labels males as females): 6

Recall is high (captures more actual females), but precision is lower (more false alarms).

Threshold 0.5 50%

The model is stricter.

This translates to a light drop in TP (misses 1 more female).

Fewer FPs which is an improvement in precision.

Fewer false alarms (better for user trust).

Threshold 0.8

The model needs 80% confidence to predict “Female”.

TP/FN stays the same (model is already very sure about those 36 females).

FP drops (high precision), with fewer mistakes calling someone Female.

As threshold increases, the model becomes more conservative in predicting “Female”, which reduces false positives (males wrongly predicted as females)

Slightly reduces true positives or keeps them the same, increases precision and may decrease recall

The higher the threshold, the more accurate the model is in it’s predictions.

3.Create a table showing—for each of the three thresholds—the accuracy, precision, recall, and F1 scores.

Threshold is the cutoff value for deciding if a prediction is positive (e.g., predicting “female”). Accuracy is the overall proportion of correct predictions (both positive and negative). Precision is all instances predicted as positive (female), how many were actually positive? Recall (also called Sensitivity) is all actual positive cases (females), how many did the model correctly identify? F1 Score is the harmonic mean of precision and recall.

```r
# Function to compute classification metrics at a given threshold
get_metrics <- function(data, threshold) {
  data <- data %>%
    mutate(
      predicted = ifelse(.pred_female >= threshold, 1, 0),
      actual = ifelse(sex == "female", 1, 0)
    )

  # Confusion matrix elements
  TP <- sum(data$predicted == 1 & data$actual == 1)
  FP <- sum(data$predicted == 1 & data$actual == 0)
  TN <- sum(data$predicted == 0 & data$actual == 0)
  FN <- sum(data$predicted == 0 & data$actual == 1)

  # Metrics
  accuracy <- (TP + TN) / (TP + TN + FP + FN)
  precision <- ifelse((TP + FP) == 0, NA, TP / (TP + FP))
  recall <- ifelse((TP + FN) == 0, NA, TP / (TP + FN))
  f1 <- ifelse(is.na(precision) | is.na(recall) | (precision + recall == 0), NA,
               2 * (precision * recall) / (precision + recall))

  return(data.frame(
    Threshold = threshold,
    Accuracy = round(accuracy, 4),
    Precision = round(precision, 4),
    Recall = round(recall, 4),
    F1_Score = round(f1, 4)
  ))
}

# List of thresholds
thresholds <- c(0.2, 0.5, 0.8)

# Combine results into a summary table
metrics_table <- do.call(rbind, lapply(thresholds, function(t) get_metrics(peng_data, t)))

# Print table
print(metrics_table)
```

```
##   Threshold Accuracy Precision Recall F1_Score
## 1       0.2   0.9140    0.8605 0.9487   0.9024
## 2       0.5   0.9355    0.9231 0.9231   0.9231
## 3       0.8   0.9462    0.9474 0.9231   0.9351
```

At low threshold (0.2): High recall, lower precision → model predicts many positives, catching almost all true positives but with more false positives.

At medium threshold (0.5): Balanced precision and recall, good overall performance.

At high threshold (0.8): High precision, slightly lower recall → model is conservative, predicting fewer positives but with high confidence.

4.Provide at least one example use case where (a) an 0.2 scored probability threshold would be preferable, and (b) an 0.8 scored probability threshold would be preferable.

a. Use Case for a 0.2 Threshold.

Is preferable when False Negatives are costly, and it’s better to “cast a wide net.” For example medical screening for rare diseases or pandemics. The priority in such case would be to Identify patients who may have a rare or contagious disease.

With this threshold many potential cases as possible are identified, even if that means getting some false positives. Missing a sick patient (false negative) could result in no treatment and more severe consequences.

Follow up testing could then eliminate false positives and we would still have erred on the side of caution.

b. Use Case for a 0.8 Threshold.

When False Positives are costly precision is prioritized, and you only want to act when very confident. When false positives are costly, we prioritize high precision (only act when very confident), even if it means missing some true positives (lower recall).

In a terminal diagnosis case the problem avoided can be automatically alerting patients they may have cancer based on test results.

You only want to notify patients if you’re highly confident the model is right.

Telling someone they might have cancer when they don’t can cause extreme anxiety, emotional distress, and may lead to unnecessary tests or biopsies.

False negatives while still serious, might be caught in follow-up screenings or doctor reviews, depending on the medical workflow.

So, it’s better to only notify the patient when confidence is very high, and have uncertain cases escalated to a doctor for review.

In banking automatically freezing accounts on suspected fraud can harm customer trust and service. A quick text to check in with the client can be advised in such case to verify.

Accept missing a few frauds (false negatives) if it avoids annoying many real users.

A high threshold = high precision, but possibly lower recall.