Analysis of Columbia Photographic Images Using Neural Networks

Timothy Ogbonnaya
219398221
Math 3333

Abstract

This project implements a neural network approach for classifying images from the Columbia Photographic Images dataset into different categories. Building upon the basic logistic regression provided in the sample code, I developed a multi-layer perceptron (MLP) neural network that leverages spatial information through image partitioning. I used a 10×10 grid decomposition of each image and then extracted 100 features. By doing this, I achieved a worse performance in terms of accuracy, specificity, and sensitivity than that of logistic regression (the results at the end of this report verify this claim).

Introduction

Because they can learn hierarchical representations from pixel data, neural networks have become extremely effective tools for image classification. In order to differentiate between indoor and outdoor categories, this study investigates their application to the "Columbia Photographic Images" dataset, which was provided by Professor Xin Gao. The neural network method has many benefits, such as:

Ability to learn from high-dimensional input (100 features vs 3)
In contrast to logistic regression, which only uses three median RGB values, the neural network method can process 100 input characteristics (100 grid cells).

The model can maintain spatial links between various image parts because of this extended input space.

Capacity to model non-linear relationships
Unlike linear models, the neural network learns complex decision boundaries through multiple non-linear transformations

Automatic feature combination and hierarchy learning
The hidden layers provide feature representations that are more complex over time.

Better handling of complex patterns in visual data
Because the network naturally manages the variability in natural settings and can adapt to different lighting conditions and angles.

This report details the implementation of a neural network classifier and compares its performance against the baseline logistic regression model.

Data

The dataset consists of 800 photographic images from the "Columbia Images" file as well as the information of the details of those pictures on the "PhotoMetaData" dataset. These images were captured by the authors using Canon 10D and Nikon D70 SLR cameras in New York City and Boston during the summer and early autumn of 2004.

Key characteristics:
Original size: 3072×2048 pixels (Canon) or 3008×2000 pixels (Nikon)
How they were saved: Stored in both RAW and JPEG formats
lighting conditions: Diverse lighting conditions such as indoor, outdoor-day, indoor-light, etc.
Content categories: natural and artificial objects

Figure 1: Examples of the different categories/lightings from the Image set provided

Methodology

A 6-step procedure on R carefully constructed the Neural Network method I used. These steps are as follows:

1. **Feature Extraction (10x10 Grids from Each Image)**
2. **Load Metadata and Initialize Variables**
3. **Extract Grid Features from Images**
4. **Train/Test Split**

5. **Train Neural Network**
6. **Evaluate Performance**

Each step is explained in further detail, with examples to help visualize the process below.

Feature Extraction

First, each image is resized to 100 x 100 pixels and converted to grayscale. After that, it is then split into a 10 x 10 grid (i.e., 100 blocks). Now, for each block, the mean pixel intensity is calculated. This results in **100 features per image** representing local brightness levels.

Figure 2: This is the transformation from a regular picture provided by the columbiaImages folder to a gray-scaled version

This is important because, instead of reducing each image to a single value per color (as in logistic regression), we retain **spatial texture and intensity information** that helps a model distinguish between patterns typical of outdoor vs. indoor scenes.

In this case, I saw that although logistic regression is easy to implement and interpret, it lacks the rigorous length that a neural network takes to ensure accuracy. However, in this particular case, the rigorousness did not ensure that the model was more accurate than the regression model.

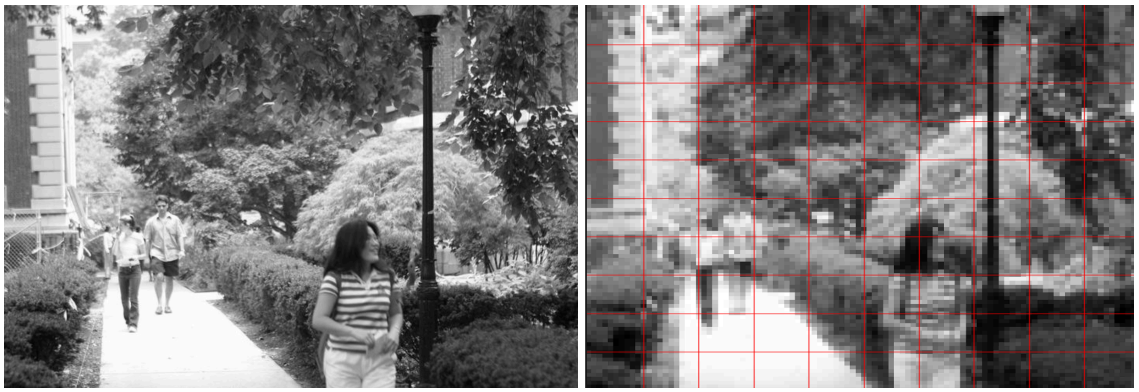


Figure 3: Conversion from just grayscale to 10 x 10 grid grayscale.

Load Metadata and Initialize Variables

In this step, I read the CSV file (photoMetaData.csv), which contains image names and categories. Labels are initialized: 1 for outdoor-day images and 0 for all other types. This is how we categorize the images.
This step is particularly important because it is the building block of addressing the classification problem- Whether an image is taken outdoors or not.

	A	B	C	D	E	F
1	name	category	camera	location	photographer	
2	CRW_4786	outdoor-d	canon 10D	new york	martin	
3	CRW_4787	outdoor-d	canon 10D	new york	martin	
4	CRW_4788	outdoor-d	canon 10D	new york	martin	
5	CRW_4789	outdoor-d	canon 10D	new york	martin	
6	CRW_4790	outdoor-d	canon 10D	new york	martin	
7	CRW_4791	outdoor-d	canon 10D	new york	martin	
8	CRW_4792	outdoor-d	canon 10D	new york	martin	
9	CRW_4794	outdoor-d	canon 10D	new york	martin	
10	CRW_4795	outdoor-d	canon 10D	new york	martin	
11	CRW_4797	outdoor-d	canon 10D	new york	martin	
12	CRW_4798	outdoor-d	canon 10D	new york	martin	
13	CRW_4799	outdoor-d	canon 10D	new york	martin	
14	CRW_4800	outdoor-d	canon 10D	new york	martin	
15	CRW_4801	outdoor-d	canon 10D	new york	martin	
16	CRW_4803	outdoor-d	canon 10D	new york	martin	

Figure 4: A snippet of the data provided from 'photoMetaData.csv'

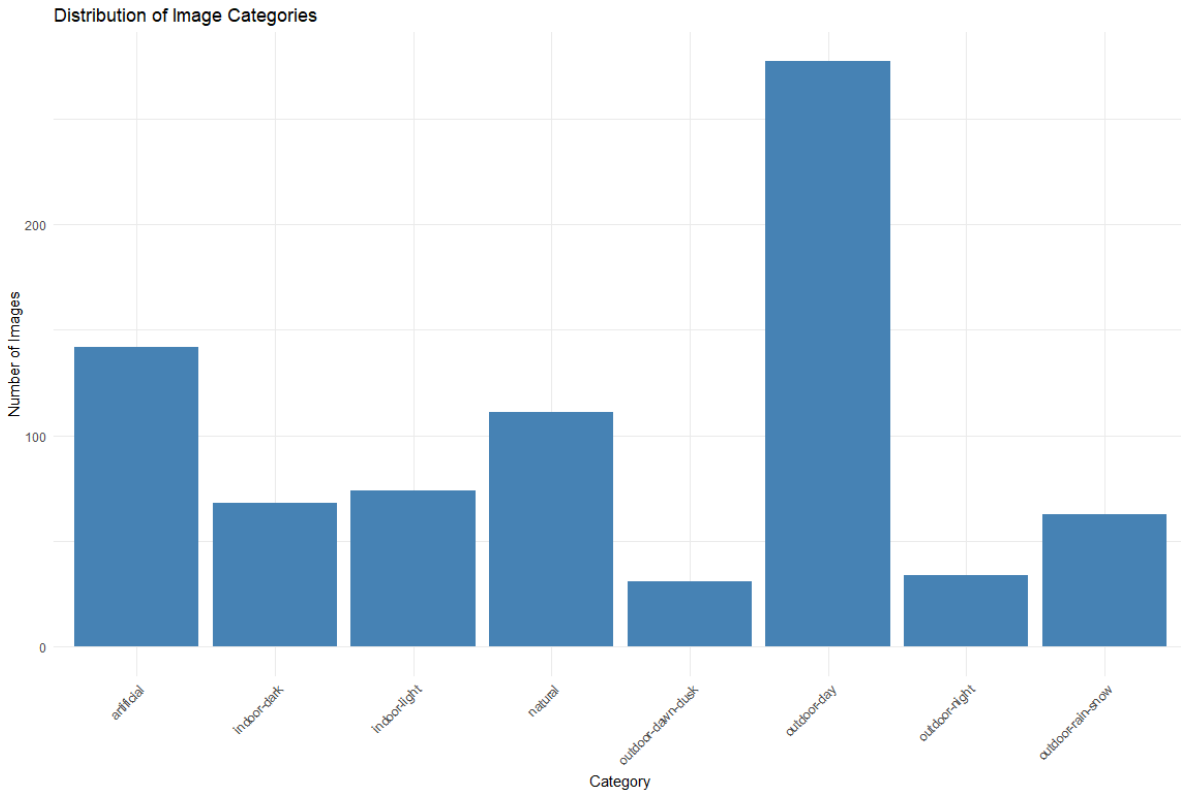


Figure 5: A bar chart showing the distributions of the different Image categories (e.g, Artificial, Indoor-dark, Outdoor-day, etc.)

Extract Grid Features from Images

Now, in this step, I merge the previous two steps by looping through all images, opening each JPEG, and converting it to grayscale, but this time, we extract the 100 grid-based mean intensity features.
This is important because we transform each raw image into numerical vectors suitable for our Neural Network model training.

Train/Test Split

I divided the dataset into 70% training and 30% testing sets at random in this step. This guarantees that the model is tested on unseen data and learns from a subset of the data. Because it avoids overfitting and permits equitable performance rating, this stage is very crucial.

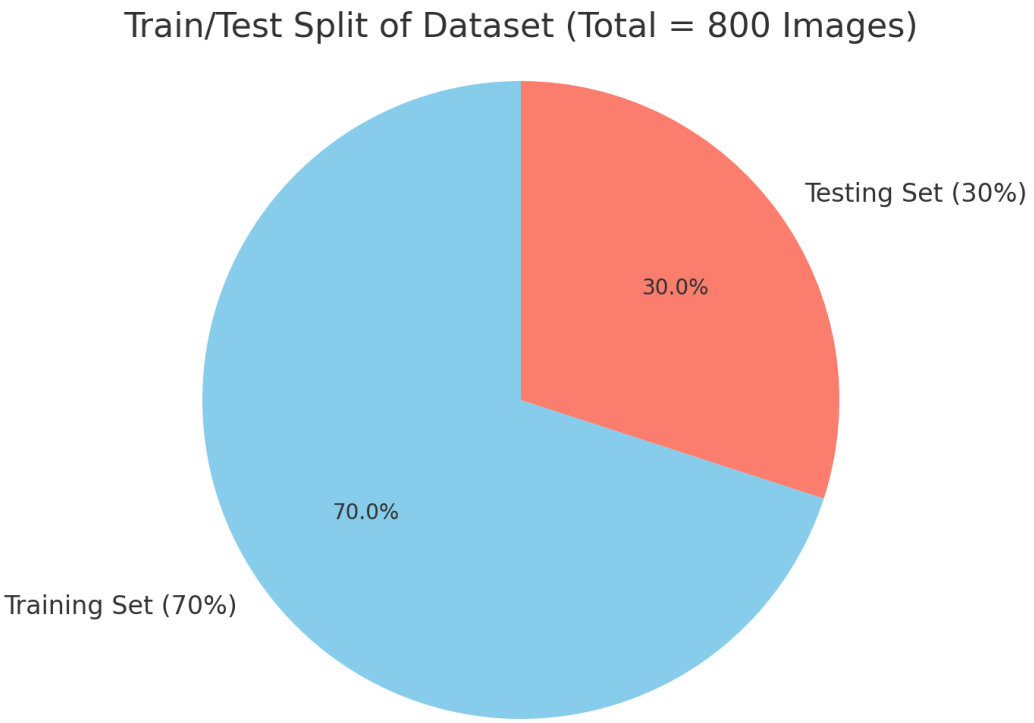


Figure 6: A visual representation showing the split explained above

Train Neural Network

In this step, we train a neural network with:

- **100 input nodes (for the 100 features)**
- **1 hidden layer with 5 neurons**
- **Output node using a sigmoid function for binary classification**

However, I use the ‘nnet’ function over the usual ‘neuralnet’ function because I wanted to deal with a single hidden layer as opposed to the multiple hidden layers that the ‘neuralnet’ offers. And most importantly, the ‘nnet’ decay parameter helps prevent overfitting

Neural Network Training Hyperparameters

Parameter	Value
Hidden Units	5.0
Weight Decay	0.01
Max Iterations	500.0

Figure 7: A table showing all the different parameters used

Evaluate Performance

In this step, I use the testing data to generate predictions and computes:

- **Accuracy:** % correctly predicted
- **Sensitivity:** % of outdoor images correctly identified
- **Specificity:** % of non-outdoor images correctly rejected
- **Misclassification rate:** % of incorrect predictions

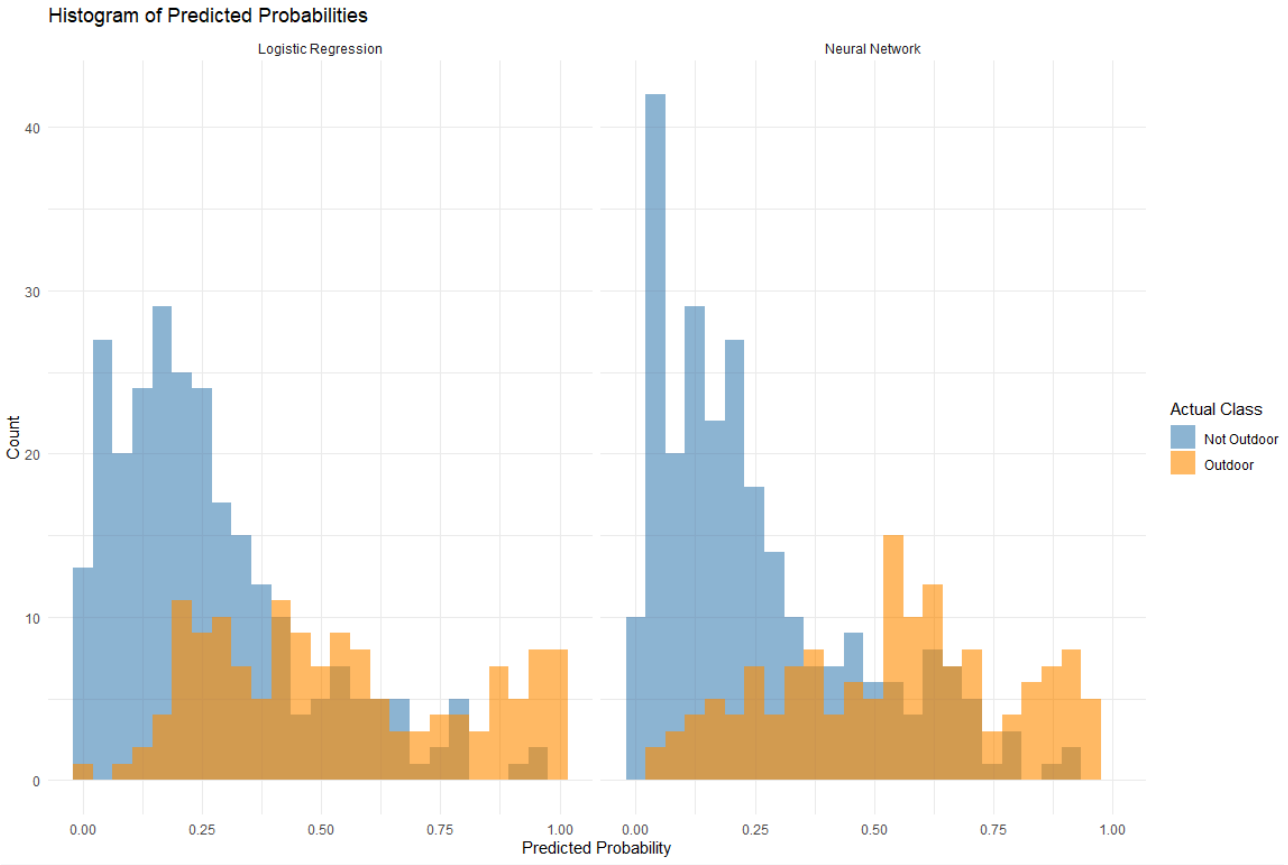
Analysis and Model

For the logistic regression model, RGB medians are extracted using apply(img, 3, median). For the neural network, grayscale conversion and grid-wise mean intensities are computed.

Logistic regression is fitted using the glm() function in R with the formula y ~ X where X contains the three RGB median values.
The neural network is trained on 100 grid-based grayscale features using 5 hidden nodes.

In the following graph, we can see the predicted probabilities shown as a histogram, which shows the classification of Outdoor pictures = 1, as opposed to indoor pictures = 0. This is interpreted as:

1. Neural Network classed a lot of indoor images (blue) nearer to 0 than Logistic Regression. This shows it could accurately determine more indoor images.
2. They both had similar classifications of Outdoor images.
3. Logistic Regression had less overlap, which made it better. Because if there's high overlap, the model is confused and makes more mistakes, and overall performance drops.



Results

Metric	Logistic Regression	Neural Network
Accuracy	73.38%	63.75%
Sensitivity	50.69%	38.37%
Specificity	86.04%	77.92%
Misclassification	26.61%	36.25%

The results indicate that Logistic Regression had a better performance in classifying outdoor/indoor images into their proper categories.

Conclusion

This study demonstrates that using detailed pixel information across a grayscale 10 x10 grid significantly decreases classification performance for detecting outdoor images, especially while using the neural network method.

The neural network, although good at computing complex structures, for this case, it offers a more inaccurate method of determining which picture was taken indoors as opposed to outdoors, and the reason for this was because of the grayscale format. It had a hard time differentiating indoor light from outdoor light (because a grayed-out picture in an indoor or outdoor setting could have similar mean intensity features).

For future work, I could improve the accuracy by using an RGB format like the one utilized for the logistic regression method, as opposed to a grayscale format, and also implement deeper architectures or convolutional neural networks for even better accuracy.

Appendix

The R script I used for the project is shown below. Remember to replace the directory from mine to yours.

```
##Author: Timothy Ogbonnaya

##Student Number: 219398221

##Final Project for Math 3333


# === Load Libraries ====

library(jpeg)

library(nnet)

library(caret)

library(pROC)


# === Load Metadata ====

meta <-
read.csv("C:/Users/timoo/OneDrive/Documents/photoMetaData.csv")

img_dir <- "C:/Users/timoo/OneDrive/Documents/columbiaImages/"


n <- nrow(meta)

y <- as.numeric(meta$category == "outdoor-day")  # Outcome: 1 =
outdoor, 0 = not


# === Feature Extraction Function ====

extract_grid_means <- function(path, size = 100, grids = 10) {

  img <- readJPEG(path)
```

```
  gray <- 0.2989 * img[,,1] + 0.5870 * img[,,2] + 0.1140 * img[,,3]
# Grayscale

  dim_grid <- size / grids

  resized <- matrix(gray[1:size, 1:size], nrow = size, ncol = size)

  features <- c()

  for (i in 0:(grids - 1)) {

    for (j in 0:(grids - 1)) {

      block <- resized[(i * dim_grid + 1):((i + 1) * dim_grid),

                       (j * dim_grid + 1):((j + 1) * dim_grid)]

      features <- c(features, mean(block, na.rm = TRUE))

    }

  }

  return(features)

}


# === Extract Features from All Images ====

X <- matrix(NA, nrow = n, ncol = 100)

for (i in 1:n) {

  path <- paste0(img_dir, meta$name[i])

  if (file.exists(path)) {

    cat(sprintf("Processing %03d/%03d: %s\n", i, n, meta$name[i]))

    X[i, ] <- tryCatch(extract_grid_means(path), error = function(e)
rep(NA, 100))

  } else {

    cat(sprintf("Missing file: %s\n", meta$name[i]))

  }

}
```

```
# === Remove Rows with All NAs (Completely Invalid Images) ====

valid_rows <- complete.cases(X)

X <- X[valid_rows, ]

y <- y[valid_rows]


# === Train/Test Split ===

set.seed(123)

train_idx <- sample(1:nrow(X), size = 0.7 * nrow(X))

test_idx <- setdiff(1:nrow(X), train_idx)


X_train <- X[train_idx, ]

X_test <- X[test_idx, ]

y_train <- y[train_idx]

y_test <- y[test_idx]


# === Mean Imputation for Any Remaining NA ====

for (i in 1:ncol(X_train)) {

  X_train[is.na(X_train[, i]), i] <- mean(X_train[, i], na.rm =
TRUE)

}

for (i in 1:ncol(X_test)) {

  X_test[is.na(X_test[, i]), i] <- mean(X_test[, i], na.rm = TRUE)

}


# === Final NA Check (Remove Bad Rows If Still Present) ====

X_train <- X_train[complete.cases(X_train), ]

y_train <- y_train[1:nrow(X_train)]
```

```
X_test <- X_test[complete.cases(X_test), ]

y_test <- y_test[1:nrow(X_test)]


# === Train Neural Network ====

nn_model <- nnet(X_train, y_train, size = 5, maxit = 500, decay =
0.01)


# === Predict and Evaluate ===

pred_probs <- predict(nn_model, X_test, type = "raw")

pred_class <- ifelse(pred_probs > 0.5, 1, 0)


conf <- confusionMatrix(factor(pred_class), factor(y_test), positive
= "1")

print(conf)


# === Custom Summary ====

cat(sprintf("\nAccuracy: %.2f%%", mean(pred_class == y_test) * 100))

cat(sprintf("\nSensitivity: %.2f%%", conf$byClass["Sensitivity"] *
100))

cat(sprintf("\nSpecificity: %.2f%%", conf$byClass["Specificity"] *
100))

cat(sprintf("\nMisclassification Rate: %.2f%%", mean(pred_class !=
y_test) * 100))


#=== Bar Chart of distribution ====


# Load libraries

library(ggplot2)
```

```
#=== bar chart of category frequencies ====

ggplot(meta, aes(x = category)) +

  geom_bar(fill = "steelblue") +

  theme_minimal() +

  labs(title = "Distribution of Image Categories",

       x = "Category",

       y = "Number of Images") +

  theme(axis.text.x = element_text(angle = 45, hjust = 1))




#=== Comparison of the Logistic Model ====


library(jpeg)

pm <-
read.csv("C:/Users/timoo/OneDrive/Documents/photoMetaData.csv")

n <- nrow(pm)


trainFlag <- (runif(n) > 0.5)

y <- as.numeric(pm$category == "outdoor-day")


X <- matrix(NA, ncol=3, nrow=n)

for (j in 1:n) {

  img <-
readJPEG(paste0("C:/Users/timoo/OneDrive/Documents/columbiaImages/",
pm$name[j]))

  X[j,] <- apply(img,3,median)
```

```
  print(sprintf("%03d / %03d", j, n))
}

out <- glm(y ~ X, family=binomial, subset=trainFlag)
out$iter
summary(out)

pred <- 1 / (1 + exp(-1 * cbind(1,X) %*% coef(out)))
y[order(pred)]
y[!trainFlag][order(pred[!trainFlag])]

mean((as.numeric(pred > 0.5) == y)[trainFlag])
mean((as.numeric(pred > 0.5) == y)[!trainFlag])




# === Train neural network on same data (3 median RGB features)
nn_model <- nnet(X[trainFlag, ], y[trainFlag], size = 5, maxit =
500, decay = 0.01)

# === Predict probabilities for test set
nn_pred <- predict(nn_model, X[!trainFlag, ], type = "raw")
glm_pred <- pred[!trainFlag]
actual <- y[!trainFlag]

# === Combine predictions for both models
```

```
df <- data.frame(

  prob = c(glm_pred, nn_pred),

  model = rep(c("Logistic Regression", "Neural Network"), each =
length(glm_pred)),

  actual = factor(rep(actual, 2), levels = c(0, 1), labels = c("Not
Outdoor", "Outdoor"))

)



# === Plot histogram comparison

library(ggplot2)

ggplot(df, aes(x = prob, fill = actual)) +

  geom_histogram(position = "identity", bins = 25, alpha = 0.6) +

  facet_wrap(~ model) +

  scale_fill_manual(values = c("steelblue", "darkorange")) +

  labs(title = "Histogram of Predicted Probabilities",

       x = "Predicted Probability",

       y = "Count",

       fill = "Actual Class") +

  theme_minimal()



glm_pred <- pred[!trainFlag]

glm_class <- ifelse(glm_pred > 0.5, 1, 0)

actual <- y[!trainFlag]



conf_logit <- confusionMatrix(factor(glm_class), factor(actual),
positive = "1")

print(conf_logit)
```

```
# Accuracy
mean(glm_class == actual) * 100

# Misclassification Rate
mean(glm_class != actual) * 100

# Sensitivity (True Positive Rate)
conf_logit$byClass["Sensitivity"] * 100

# Specificity (True Negative Rate)
conf_logit$byClass["Specificity"] * 100
```

References

MATH 3333 Lecture Notes, York University (2025).

MATH 3333 Project Sample Code

Columbia Images, Columbia University.
