Machine Learning fairness

AUTHOR
Saadullah Khan

Social Background

The notion of machine learning The choice of the algorithm is influenced by bias during data collection, which results in the under representation of certain groups. Failure to resolve this prejudice compromises the fairness of machine learning. There are numerous applications for machine learning fairness, including the confirmation of credit cards and the recruitment of specific positions. There are numerous factors that can be considered during the hiring process, including gender, age, experience, and level of education. However, if the algorithm used to hire is solely based on whether or not an applicant has a master's degree, it will be unjust for many applicants. This is because many applicants may not have a master's degree but still meet the requirements of the position. Consequently, the ML system will be unjust. This is significant because the system favors individuals with high incomes over those with lower incomes, and it discriminates against people who cannot afford to pay for master's programs and progress. Many parameters are considered when evaluating credit card approval, including age, income, and type of employment. The algorithm is rendered unjust when credit card approval is contingent upon an individual's income. This is significant because it affects a significant number of individuals across various social classes. Specifically, individuals with low incomes will be unable to establish a credit score in order to purchase a home or vehicle in the future.

Overview of definitions used to measure ML Fairness:

The impartiality of machine learning is primarily categorized into five groups. The initial principle is statistical or demographic parity, which requires that all outputs be IID. This implies that the ML system should regard diverse demographic groups equally, resulting in comparable outcomes for all. The second option is equalized probabilities, which implies that it should perform equally well for each group in terms of accurately predicting positive and negative events. The third principle is equality of opportunity, which posits that the ML system should provide each group with an equal opportunity to accomplish positive outcomes when they are well-deserved. Predictive parity is the fourth concept. This implies that the algorithm must be equally precise in its ability to predict favorable outcomes for each group. The subsequent phase is calibration, which is the process of ensuring that 70% of the cases in a specific group result in a positive outcome if the system predicts a 70% probability of a favorable outcome.

The Data

The data is a prediction of loan approval. The data set comprises eight categories: age, income, employment duration, loan intent, loan grade, loan amount, interest rate, income-to-loan ratio, credit history length, and historical default. We will forecast whether the loans were paid or defaulted. The fairness metric we are examining is whether age has a substantial impact on loan default. If all other parameters are used, does this affect the payment of the loan? The data has been divided into three sections: 50% is designated for training, 30% for validation, and 20% for testing.

1. **Naive Bayes Model**

```
# Define classes and calculate priors
classes <- unique(Train_loan$loan_status)
priors <- prop.table(table(Train_loan$loan_status))
```

```r
# Identify variable types
numeric_cols <- sapply(Train_loan, is.numeric)
categorical_cols <- sapply(Train_loan, is.factor)
categorical_cols[names(Train_loan) == "loan_status"] <- FALSE  # Exclude target variable

# Create summaries for numeric features with Laplace smoothing
numeric_summary <- lapply(classes, function(cls) {
  data <- Train_loan[Train_loan$loan_status == cls, numeric_cols, drop = FALSE]
  data.frame(
    mean = sapply(data, function(x) mean(x, na.rm = TRUE)),
    sd = sapply(data, function(x) {
      sd_val <- sd(x, na.rm = TRUE)
      ifelse(is.na(sd_val) || sd_val < 1e-10, 1e-10, sd_val)  # Avoid zero sd
    })
  )
})
names(numeric_summary) <- classes

# Create frequency tables for categorical features with Laplace smoothing
categorical_summary <- lapply(classes, function(cls) {
  data <- Train_loan[Train_loan$loan_status == cls, categorical_cols, drop = FALSE]

  lapply(data, function(col) {
    all_levels <- levels(col)
    freq_table <- table(col)
    smoothed_table <- freq_table + 1

    for (lvl in all_levels) {
      if (!(lvl %in% names(smoothed_table))) {
        smoothed_table[lvl] <- 1
      }
    }

    prop.table(smoothed_table)
  })
})
names(categorical_summary) <- classes

# Naive Bayes prediction function
predict_nb <- function(obs, priors, numeric_summary, categorical_summary) {
  log_probs <- numeric(length(priors))
  names(log_probs) <- names(priors)

  for (cls in names(priors)) {
    # Start with log prior
    log_probs[cls] <- log(priors[cls])

    # Numeric predictors: Gaussian likelihood
    for (col in names(which(numeric_cols))) {
      x <- as.numeric(obs[[col]])
      if (!is.na(x)) {
        mu <- numeric_summary[[cls]][col, "mean"]
```

```
        sd <- numeric_summary[[cls]][col, "sd"]
        log_probs[cls] <- log_probs[cls] + dnorm(x, mean = mu, sd = sd, log = TRUE)
      }
    }

    # Categorical predictors: frequency-based probability
    for (col in names(which(categorical_cols))) {
      val <- as.character(obs[[col]])
      if (!is.na(val)) {
        probs_table <- categorical_summary[[cls]][[col]]
        if (val %in% names(probs_table)) {
          log_probs[cls] <- log_probs[cls] + log(probs_table[val])
        } else {
          log_probs[cls] <- log_probs[cls] + log(1 / (length(probs_table) + 1))
        }
      }
    }
  }

  return(names(which.max(log_probs)))
}

# Predict with Naive Bayes
n_test <- nrow(test_loan)
nb_preds <- character(n_test)

for (i in 1:n_test) {
  nb_preds[i] <- predict_nb(test_loan[i,], priors, numeric_summary, categorical_summary)
}

# Evaluate Naive Bayes
nb_conf_matrix <- table(Predicted = nb_preds, Actual = test_loan$loan_status)


nb_accuracy <- sum(diag(nb_conf_matrix)) / sum(nb_conf_matrix)

cat("Naive Bayes - Class priors:\n")
```

```
Naive Bayes - Class priors:
```

```
print(priors)
```

```
defaulted      Paid
0.2198895 0.7801105
```

```
cat("Predicting with Naive Bayes...\n")
```

```
Predicting with Naive Bayes...
```

```
cat("\nNaive Bayes Confusion Matrix:\n")
```

```
Naive Bayes Confusion Matrix:
```

```
print(nb_conf_matrix)
```

```
           Actual
Predicted   defaulted Paid
  defaulted       920  559
  Paid            481 4557
```

```
cat("\nNaive Bayes Accuracy:", round(nb_accuracy * 100, 2), "%\n")
```

```
Naive Bayes Accuracy: 84.04 %
```

The Naive Bayes model is operational and quite accurate on test data, as evidenced by its 84 percent accuracy and 16 percent error rate. However, the prediction could be further enhanced with additional data or access to additional training data.

2. **Linear Discriminant Analysis**

```
# Function to handle NAs in numeric data
preprocess_data <- function(data) {
  numeric_vars <- names(data)[sapply(data, is.numeric)]
  numeric_vars <- setdiff(numeric_vars, "loan_status")
  complete_indices <- complete.cases(data[, numeric_vars])
  data_clean <- data[complete_indices, ]
  return(list(data = data_clean, numeric_vars = numeric_vars))
}

# Prepare data for LDA
train_processed <- preprocess_data(Train_loan)
test_processed <- preprocess_data(test_loan)

Train_loan_clean <- train_processed$data
test_loan_clean <- test_processed$data
numeric_vars <- train_processed$numeric_vars

X_train <- Train_loan_clean[, numeric_vars]
y_train <- Train_loan_clean$loan_status
X_test <- test_loan_clean[, numeric_vars]
y_test <- test_loan_clean$loan_status

# Fit LDA model
lda_fit <- lda(y_train ~ ., data = data.frame(y_train = y_train, X_train))
lda_preds <- predict(lda_fit, newdata = data.frame(X_test))$class

# Evaluate LDA
lda_conf_matrix <- table(Predicted = lda_preds, Actual = y_test)
cat("\nLDA Confusion Matrix:\n")
```

```
LDA Confusion Matrix:
```

```
print(lda_conf_matrix)
```

```
           Actual
Predicted   defaulted Paid
  defaulted       525  255
  Paid            704 4269
```

```
lda_accuracy <- sum(diag(lda_conf_matrix)) / sum(lda_conf_matrix)
cat("\nLDA Accuracy:", round(lda_accuracy * 100, 2), "%\n")
```

```
LDA Accuracy: 83.33 %
```

This model is comparable to Naive Bayes; however, its performance is marginally inferior. Had we had access to additional training data or more data, it may have been performing more effectively.

3. **logistic regression**

```
# Prepare data for logistic regression
prepare_log_data <- function(data, response_var, reference_levels = NULL) {
  data_copy <- data

  if (!is.factor(data_copy[[response_var]])) {
    data_copy[[response_var]] <- as.factor(data_copy[[response_var]])
  }

  if (!is.null(reference_levels)) {
    data_copy[[response_var]] <- factor(data_copy[[response_var]], levels = reference_levels)
  }

  numeric_vars <- names(data_copy)[sapply(data_copy, is.numeric)]
  complete_rows <- complete.cases(data_copy[, c(numeric_vars, response_var)])
  data_clean <- data_copy[complete_rows, ]

  X <- as.matrix(data_clean[, numeric_vars])
  y <- data_clean[[response_var]]
  y_bin <- ifelse(y == levels(y)[1], 1, -1)
  X_with_intercept <- cbind(1, X)

  return(list(
    X = X_with_intercept,
    y = y_bin,
    feature_names = c("Intercept", numeric_vars),
    original_y = y,
    levels = levels(y),
    clean_data = data_clean
  ))
}

# Fit logistic regression with regularization
fit_logistic <- function(X, y, regularization = "none", lambda = 1) {
  n <- nrow(X)
  p <- ncol(X)
```

```
  w <- Variable(p)
  loss <- sum_entries(logistic(-multiply(y, X %*% w)))

  objective <- loss
  if (regularization == "ridge") {
    ridge_penalty <- lambda * sum_squares(w[2:p])
    objective <- objective + ridge_penalty
  } else if (regularization == "lasso") {
    lasso_penalty <- lambda * p_norm(w[2:p], 1)
    objective <- objective + lasso_penalty
  }

  problem <- Problem(Minimize(objective))
  result <- solve(problem)

  list(
    w = result$getValue(w),
    objective = result$value,
    status = result$status
  )
}

# Predict with logistic regression
predict_logistic <- function(X, w, type = "class", levels = c("defaulted", "paid")) {
  probs <- 1 / (1 + exp(-X %*% w))

  if (type == "class") {
    raw_pred <- ifelse(probs >= 0.5, 1, -1)
    return(ifelse(raw_pred == 1, levels[1], levels[2]))
  } else {
    return(probs)
  }
}

# Prepare data for logistic regression
train_log_data <- prepare_log_data(Train_loan, "loan_status")
test_log_data <- prepare_log_data(test_loan, "loan_status", reference_levels = levels(train_log_data$orig

# Fit logistic regression models
log_model <- fit_logistic(train_log_data$X, train_log_data$y)
ridge_model <- fit_logistic(train_log_data$X, train_log_data$y, "ridge", lambda = 1)
lasso_model <- fit_logistic(train_log_data$X, train_log_data$y, "lasso", lambda = 1)

# Make predictions
log_preds <- predict_logistic(test_log_data$X, log_model$w, levels = train_log_data$levels)
ridge_preds <- predict_logistic(test_log_data$X, ridge_model$w, levels = train_log_data$levels)
lasso_preds <- predict_logistic(test_log_data$X, lasso_model$w, levels = train_log_data$levels)

# Evaluate logistic regression models
evaluate_model <- function(y_pred, y_true, levels) {
  if (is.numeric(y_true)) {
    y_true <- ifelse(y_true == 1, levels[1], levels[2])
  }
```

```
  conf_matrix <- table(Predicted = y_pred, Actual = y_true)
  accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)

  if (length(levels) == 2) {
    TP <- sum(y_pred == levels[1] & y_true == levels[1])
    FP <- sum(y_pred == levels[1] & y_true == levels[2])
    TN <- sum(y_pred == levels[2] & y_true == levels[2])
    FN <- sum(y_pred == levels[2] & y_true == levels[1])

    precision <- ifelse(TP + FP > 0, TP / (TP + FP), 0)
    recall <- ifelse(TP + FN > 0, TP / (TP + FN), 0)
    f1 <- ifelse(precision + recall > 0, 2 * precision * recall / (precision + recall), 0)

    return(list(
      confusion_matrix = conf_matrix,
      accuracy = accuracy,
      precision = precision,
      recall = recall,
      f1 = f1
    ))
  } else {
    return(list(
      confusion_matrix = conf_matrix,
      accuracy = accuracy
    ))
  }
}

# Evaluate logistic regression models
log_eval <- evaluate_model(log_preds, test_log_data$y, train_log_data$levels)
ridge_eval <- evaluate_model(ridge_preds, test_log_data$y, train_log_data$levels)
lasso_eval <- evaluate_model(lasso_preds, test_log_data$y, train_log_data$levels)

cat("\nLogistic Regression Confusion Matrix:\n")
```

```
Logistic Regression Confusion Matrix:
```

```
print(log_eval$confusion_matrix)
```

```
           Actual
Predicted   defaulted Paid
  defaulted       501  231
  Paid            728 4293
```

```
cat("Accuracy:", round(log_eval$accuracy * 100, 2), "%\n")
```

```
Accuracy: 83.33 %
```

```
cat("F1 Score:", round(log_eval$f1, 4), "\n")
```

```
F1 Score: 0.511
```

```
cat("\nLasso Regression Confusion Matrix:\n")
```

```
Lasso Regression Confusion Matrix:
```

```
print(lasso_eval$confusion_matrix)
```

```
           Actual
Predicted   defaulted Paid
  defaulted       502  228
  Paid            727 4296
```

```
cat("Accuracy:", round(lasso_eval$accuracy * 100, 2), "%\n")
```

```
Accuracy: 83.4 %
```

```
cat("F1 Score:", round(lasso_eval$f1, 4), "\n")
```

```
F1 Score: 0.5125
```

This model boasts accuracy that is comparable to that of the LDA and a f1 score of 0.51, indicating that it operates at an average level of efficiency. Nevertheless, there is room for improvement. The only distinction is that the client defaulted on the loan, even though logistic regression predicted a higher number of paid loans. In this context, the bank incurs substantial losses since 728 individuals defaulted on their loans, even though the model indicates that they repaid the loans.

4. **logistic Ridge Model**

```
Ridge Regression Confusion Matrix:

           Actual
Predicted   defaulted Paid
  defaulted       481  211
  Paid            748 4313

Accuracy: 83.33 %

F1 Score: 0.5008
```

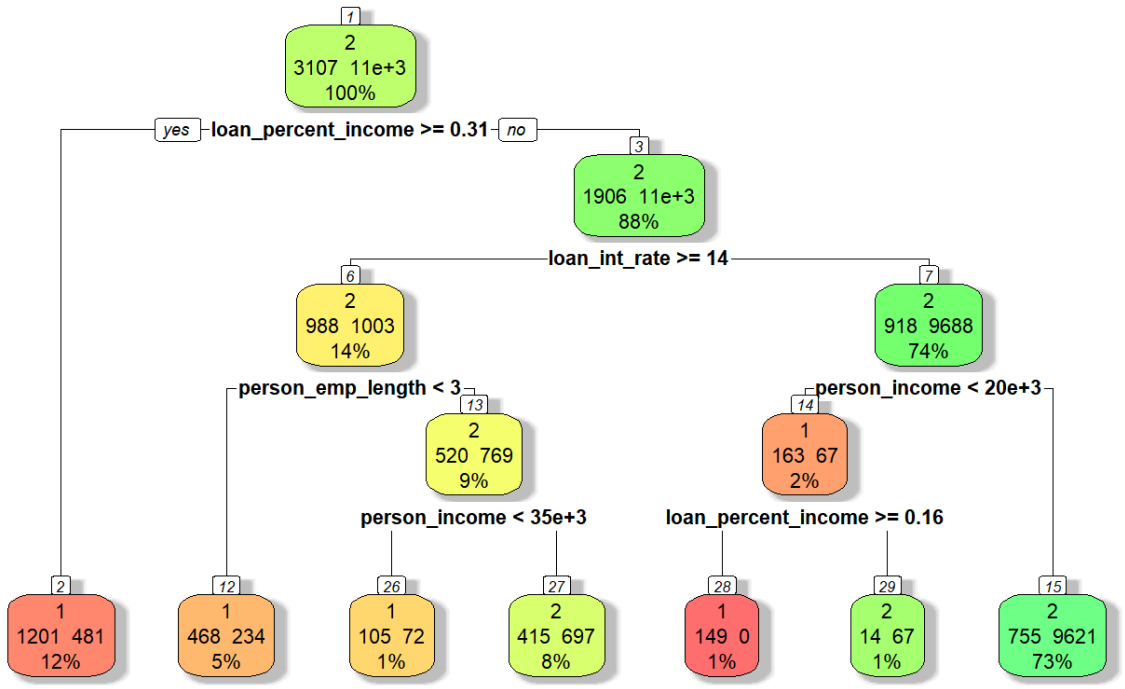
5. **logistic Lasso Model**

```
Lasso Regression Confusion Matrix:

           Actual
Predicted   defaulted Paid
  defaulted       502  228
  Paid            727 4296

Accuracy: 83.4 %
```

```
F1 Score: 0.5125
```

6. **Decision Tree**

```
# Fix the loan_status typo
Train_loan$loan_status <- ifelse(Train_loan$loan_status == "defulated", "defaulted", Train_loan$loan_
test_loan$loan_status <- ifelse(test_loan$loan_status == "defulated", "defaulted", test_loan$loan_sta

# Ensure loan_status is a factor
Train_loan$loan_status <- as.factor(Train_loan$loan_status)
test_loan$loan_status <- factor(test_loan$loan_status, levels = levels(Train_loan$loan_status))

# Remove NA values
Train_loan <- na.omit(Train_loan)
test_loan <- na.omit(test_loan)

# Extract numeric predictors
numeric_vars <- names(Train_loan)[sapply(Train_loan, is.numeric)]
X_train <- Train_loan[, numeric_vars]
y_train <- Train_loan$loan_status

# Train decision tree model
tree_model <- rpart(loan_status ~ ., data = Train_loan[, c(numeric_vars, "loan_status")],
                    method = "class",
                    control = rpart.control(minsplit = 20, minbucket = 7, cp = 0.01))

# Plot the decision tree
rpart.plot(tree_model,
           extra = 101,
           box.palette = "RdYlGn",
           shadow.col = "gray",
           nn = TRUE,
           fallen.leaves = TRUE,
           main = "Decision Tree for Loan Status Prediction")
```




```
# Test data preparation
X_test <- test_loan[, numeric_vars]
y_test <- test_loan$loan_status

# Predictions using decision tree
tree_preds <- predict(tree_model, X_test, type = "class")
tree_probs <- predict(tree_model, X_test, type = "prob")

# Confusion matrix for decision tree
tree_conf <- table(Predicted = tree_preds, Actual = y_test)
cat("\nDecision Tree Confusion Matrix:\n")
```

```
Decision Tree Confusion Matrix:
```

```
print(tree_conf)
```

```
         Actual
Predicted    1    2
        1  752  310
        2  477 4214
```

```
# Calculate accuracy for decision tree
tree_accuracy <- sum(diag(tree_conf)) / sum(tree_conf)
cat("\nDecision Tree Accuracy:", round(tree_accuracy * 100, 2), "%\n")
```

```
Decision Tree Accuracy: 86.32 %
```

```r
# Calculate precision, recall, and F1 for defaulted loans
if ("defaulted" %in% levels(y_test)) {
  TP <- sum(tree_preds == "defaulted" & y_test == "defaulted")
  FP <- sum(tree_preds == "defaulted" & y_test != "defaulted")
  FN <- sum(tree_preds != "defaulted" & y_test == "defaulted")

  precision <- ifelse(TP + FP > 0, TP / (TP + FP), 0)
  recall <- ifelse(TP + FN > 0, TP / (TP + FN), 0)
  f1 <- ifelse(precision + recall > 0, 2 * precision * recall / (precision + recall), 0)

  cat("\nDecision Tree Performance for 'defaulted' class:\n")
  cat("Precision:", round(precision, 4), "\n")
  cat("Recall:", round(recall, 4), "\n")
  cat("F1 Score:", round(f1, 4), "\n")
}
```

The decision tree is the most effective implementation, as it consistently makes the most accurate predictions and has a significantly lower rate of false paid than all previous models. The critical factor to consider when determining whether or not an individual will pay is their loan-to-income ratio, as indicated by the decision tree. This model would be the most precise if the bank used it, as it makes fewer errors.

7. **Ensamble/Fairstake**

   The fairstake, which includes all of the foundation models from earlier, is subject to the constraint that age can be discriminated against. We employ logistic regression with L2 regularization.

```r
# Function to generate base model predictions for stacking
get_base_predictions <- function(train_data, test_data, response_var) {
  # Clean data for modeling
  train_clean <- na.omit(train_data)
  test_clean <- na.omit(test_data)

  # Ensure response variable is a factor with consistent levels
  train_clean[[response_var]] <- as.factor(train_clean[[response_var]])
  if (!is.factor(test_clean[[response_var]])) {
    test_clean[[response_var]] <- factor(test_clean[[response_var]], levels = levels(train_clean[[res
  }

  # Extract numeric predictors
  numeric_vars <- names(train_clean)[sapply(train_clean, is.numeric)]
  categorical_cols <- sapply(train_clean, is.factor)
  categorical_cols[names(train_clean) == response_var] <- FALSE

  # Initialize output matrix to store predictions
  test_preds <- matrix(NA, nrow = nrow(test_clean), ncol = 5)
  colnames(test_preds) <- c("nb_pred", "lda_pred", "log_pred", "ridge_pred", "tree_pred")

  # 1. Naive Bayes Model
  # Calculate priors
  classes <- unique(train_clean[[response_var]])
```

```r
priors <- prop.table(table(train_clean[[response_var]]))

# Create summaries for numeric features with Laplace smoothing
numeric_summary <- lapply(classes, function(cls) {
  data <- train_clean[train_clean[[response_var]] == cls, numeric_vars, drop = FALSE]
  data.frame(
    mean = sapply(data, function(x) mean(x, na.rm = TRUE)),
    sd = sapply(data, function(x) {
      sd_val <- sd(x, na.rm = TRUE)
      ifelse(is.na(sd_val) || sd_val < 1e-10, 1e-10, sd_val)  # Avoid zero sd
    })
  )
})
names(numeric_summary) <- classes

# Create frequency tables for categorical features with Laplace smoothing
categorical_summary <- lapply(classes, function(cls) {
  data <- train_clean[train_clean[[response_var]] == cls, categorical_cols, drop = FALSE]

  lapply(data, function(col) {
    all_levels <- levels(col)
    freq_table <- table(col)
    smoothed_table <- freq_table + 1

    for (lvl in all_levels) {
      if (!(lvl %in% names(smoothed_table))) {
        smoothed_table[lvl] <- 1
      }
    }

    prop.table(smoothed_table)
  })
})
names(categorical_summary) <- classes

# Naive Bayes prediction function
predict_nb <- function(obs, priors, numeric_summary, categorical_summary) {
  log_probs <- numeric(length(priors))
  names(log_probs) <- names(priors)

  for (cls in names(priors)) {
    # Start with log prior
    log_probs[cls] <- log(priors[cls])

    # Numeric predictors: Gaussian likelihood
    for (col in names(which(numeric_vars %in% names(obs)))) {
      x <- as.numeric(obs[[col]])
      if (!is.na(x)) {
        mu <- numeric_summary[[cls]][col, "mean"]
        sd <- numeric_summary[[cls]][col, "sd"]
        log_probs[cls] <- log_probs[cls] + dnorm(x, mean = mu, sd = sd, log = TRUE)
      }
    }
```

```
    # Categorical predictors
    for (col in names(which(categorical_cols))) {
      if (col %in% names(obs)) {
        val <- as.character(obs[[col]])
        if (!is.na(val)) {
          probs_table <- categorical_summary[[cls]][[col]]
          if (val %in% names(probs_table)) {
            log_probs[cls] <- log_probs[cls] + log(probs_table[val])
          } else {
            log_probs[cls] <- log_probs[cls] + log(1 / (length(probs_table) + 1))
          }
        }
      }
    }
  }

  # Convert class to binary for stacking (1 for "defaulted", 0 for "paid")
  max_class <- names(which.max(log_probs))
  return(ifelse(max_class == "defaulted", 1, 0))
}

# Make Naive Bayes predictions
for (i in 1:nrow(test_clean)) {
  test_preds[i, "nb_pred"] <- predict_nb(test_clean[i,], priors, numeric_summary, categorical_summa
}

# 2. LDA Model
lda_fit <- lda(formula(paste(response_var, "~ .")), data = train_clean[, c(numeric_vars, response_v
lda_pred_obj <- predict(lda_fit, newdata = test_clean)
# Convert class to binary (1 for "defaulted", 0 for "paid")
test_preds[, "lda_pred"] <- ifelse(lda_pred_obj$class == "defaulted", 1, 0)

# 3. Logistic Regression
# Prepare data for logistic regression
prepare_log_data <- function(data, response_var, reference_levels = NULL) {
  data_copy <- data

  if (!is.factor(data_copy[[response_var]])) {
    data_copy[[response_var]] <- as.factor(data_copy[[response_var]])
  }

  if (!is.null(reference_levels)) {
    data_copy[[response_var]] <- factor(data_copy[[response_var]], levels = reference_levels)
  }

  numeric_vars <- names(data_copy)[sapply(data_copy, is.numeric)]
  complete_rows <- complete.cases(data_copy[, c(numeric_vars, response_var)])
  data_clean <- data_copy[complete_rows, ]

  X <- as.matrix(data_clean[, numeric_vars])
  y <- data_clean[[response_var]]
  y_bin <- ifelse(y == levels(y)[1], 1, -1)
  X_with_intercept <- cbind(1, X)
```

```r
  return(list(
    X = X_with_intercept,
    y = y_bin,
    feature_names = c("Intercept", numeric_vars),
    original_y = y,
    levels = levels(y),
    clean_data = data_clean
  ))
}

# Fit logistic regression with regularization
fit_logistic <- function(X, y, regularization = "none", lambda = 1) {
  n <- nrow(X)
  p <- ncol(X)

  w <- Variable(p)
  loss <- sum_entries(logistic(-multiply(y, X %*% w)))

  objective <- loss
  if (regularization == "ridge") {
    ridge_penalty <- lambda * sum_squares(w[2:p])
    objective <- objective + ridge_penalty
  } else if (regularization == "lasso") {
    lasso_penalty <- lambda * p_norm(w[2:p], 1)
    objective <- objective + lasso_penalty
  }

  problem <- Problem(Minimize(objective))
  result <- solve(problem)

  list(
    w = result$getValue(w),
    objective = result$value,
    status = result$status
  )
}

# Predict with logistic regression
predict_logistic <- function(X, w) {
  probs <- 1 / (1 + exp(-X %*% w))
  return(ifelse(probs >= 0.5, 1, 0))
}

train_log_data <- prepare_log_data(train_clean, response_var)
test_log_data <- prepare_log_data(test_clean, response_var, reference_levels = levels(train_log_dat

# Regular logistic regression
log_model <- fit_logistic(train_log_data$X, train_log_data$y)
test_preds[, "log_pred"] <- predict_logistic(test_log_data$X, log_model$w)

# Ridge logistic regression
ridge_model <- fit_logistic(train_log_data$X, train_log_data$y, "ridge", lambda = 1)
test_preds[, "ridge_pred"] <- predict_logistic(test_log_data$X, ridge_model$w)
```

```r
  # 4. Decision Tree
  tree_model <- rpart(formula(paste(response_var, "~ .")),
                      data = train_clean[, c(numeric_vars, response_var)],
                      method = "class",
                      control = rpart.control(minsplit = 20, minbucket = 7, cp = 0.01))

  tree_probs <- predict(tree_model, test_clean, type = "prob")
  test_preds[, "tree_pred"] <- ifelse(tree_probs[, "defaulted"] >= 0.5, 1, 0)

  # Convert to data frame and add actual outcomes
  result_df <- data.frame(test_preds)
  result_df$actual <- ifelse(test_clean[[response_var]] == "defaulted", 1, 0)

  return(result_df)
}

# Function to create stacked model using cross-validation
create_stacked_model <- function(data, response_var, k = 1) {
  set.seed(42)

  # Create k folds for cross-validation
  folds <- createFolds(data[[response_var]], k = k, list = TRUE, returnTrain = FALSE)
  meta_data <- data.frame()

  # Run k-fold CV to create meta-features
  for (i in 1:k) {
    test_idx <- folds[[i]]
    train_idx <- setdiff(1:nrow(data), test_idx)

    cv_train <- data[train_idx, ]
    cv_test <- data[test_idx, ]

    fold_preds <- get_base_predictions(cv_train, cv_test, response_var)
    meta_data <- rbind(meta_data, fold_preds)
  }

  # Train meta-model (logistic regression with logistic loss) on all meta-features
  X_meta <- as.matrix(meta_data[, !colnames(meta_data) %in% "actual"])
  y_meta <- ifelse(meta_data$actual == 1, 1, -1)  # Convert to -1/1 format for CVXR

  # Add intercept
  X_meta_with_intercept <- cbind(1, X_meta)

  # Use CVXR to solve logistic regression with logistic loss
  p <- ncol(X_meta_with_intercept)
  w <- Variable(p)

  # Logistic loss function
  logistic_loss <- sum_entries(logistic(-multiply(y_meta, X_meta_with_intercept %*% w)))

  # Add L2 regularization
  lambda <- 0.01
  reg_term <- lambda * sum_squares(w[2:p])
```

```r
  # Define and solve the optimization problem
  objective <- logistic_loss + reg_term
  problem <- Problem(Minimize(objective))
  result <- solve(problem)

  meta_model <- list(
    coefficients = result$getValue(w),
    feature_names = c("intercept", colnames(X_meta))
  )

  return(meta_model)
}

# Predict using the stacked model
predict_stacked <- function(test_data, train_data, meta_model, response_var) {
  # Get base model predictions
  base_preds <- get_base_predictions(train_data, test_data, response_var)

  # Extract features and add intercept
  X_test <- as.matrix(base_preds[, !colnames(base_preds) %in% "actual"])
  X_test_with_intercept <- cbind(1, X_test)

  # Make predictions using meta-model
  logits <- X_test_with_intercept %*% meta_model$coefficients
  probs <- 1 / (1 + exp(-logits))
  preds <- ifelse(probs >= 0.5, "defaulted", "paid")

  # Return results
  results <- data.frame(
    Actual = ifelse(base_preds$actual == 1, "defaulted", "paid"),
    Predicted = preds,
    Probability = as.vector(probs)
  )

  return(results)
}

# Evaluate model performance
evaluate_classifier <- function(actual, predicted) {
  conf_matrix <- table(Predicted = predicted, Actual = actual)
  accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)

  # Calculate precision, recall, F1 for "defaulted" class
  tp <- sum(predicted == "defaulted" & actual == "defaulted")
  fp <- sum(predicted == "defaulted" & actual == "paid")
  fn <- sum(predicted == "paid" & actual == "defaulted")

  precision <- ifelse(tp + fp > 0, tp / (tp + fp), 0)
  recall <- ifelse(tp + fn > 0, tp / (tp + fn), 0)
  f1 <- ifelse(precision + recall > 0, 2 * precision * recall / (precision + recall), 0)

  return(list(
    confusion_matrix = conf_matrix,
    accuracy = accuracy,
```

```
    precision = precision,
    recall = recall,
    f1 = f1
  ))
}

# Replace the dataset loading and preparation with the existing data
loan <- read.csv("credit_risk_dataset.csv")
loan$loan_status <- ifelse(loan$loan_status == "1", "defaulted", "Paid")

# Create train/validation/test split
set.seed(9890)
prop <- c(train=.5, valid=.3, test=.2)
shuf <- sample(cut(seq(nrow(loan)), nrow(loan)*cumsum(c(0,prop)), labels=names(prop)))
splits <- split(loan, shuf)

Train_loan <- splits$train
valid_loan <- splits$valid
test_loan <- splits$test

# Ensure consistent data types
prep_data <- function(data) {
  if("loan_status" %in% names(data)) {
    data$loan_status <- as.factor(data$loan_status)
  }

  char_cols <- sapply(data, is.character)
  data[char_cols] <- lapply(data[char_cols], as.factor)

  return(data)
}

Train_loan <- prep_data(Train_loan)
valid_loan <- prep_data(valid_loan)
test_loan <- prep_data(test_loan)

# Lowercase the 'Paid' to match with model expectations
Train_loan$loan_status <- as.character(Train_loan$loan_status)
Train_loan$loan_status <- ifelse(Train_loan$loan_status == "Paid", "paid", Train_loan$loan_status)
Train_loan$loan_status <- as.factor(Train_loan$loan_status)

valid_loan$loan_status <- as.character(valid_loan$loan_status)
valid_loan$loan_status <- ifelse(valid_loan$loan_status == "Paid", "paid", valid_loan$loan_status)
valid_loan$loan_status <- as.factor(valid_loan$loan_status)

test_loan$loan_status <- as.character(test_loan$loan_status)
test_loan$loan_status <- ifelse(test_loan$loan_status == "Paid", "paid", test_loan$loan_status)
test_loan$loan_status <- as.factor(test_loan$loan_status)

# Train stacked model using cross-validation on training data
stacked_model <- create_stacked_model(Train_loan, "loan_status", k = 5)

# Show meta-model coefficients
cat("\nMeta-model coefficients:\n")
```

Meta-model coefficients:

```
coef_table <- data.frame(
  Feature = stacked_model$feature_names,
  Coefficient = stacked_model$coefficients
)
print(coef_table)
```

```
     Feature Coefficient
1  intercept  -2.5218936
2    nb_pred   1.9243834
3   lda_pred   0.3232296
4   log_pred   0.6282114
5 ridge_pred  -0.1277706
6  tree_pred   2.5506173
```

```
# Make predictions on validation set
cat("\nMaking predictions on validation set...\n")
```

Making predictions on validation set...

```
valid_preds <- predict_stacked(valid_loan, Train_loan, stacked_model, "loan_status")
valid_eval <- evaluate_classifier(valid_preds$Actual, valid_preds$Predicted)

cat("\nValidation Set Performance:\n")
```

Validation Set Performance:

```
cat("Confusion Matrix:\n")
```

Confusion Matrix:

```
print(valid_eval$confusion_matrix)
```

```
           Actual
Predicted   defaulted paid
  defaulted      1168  488
  paid            699 6251
```

```
cat("Accuracy:", round(valid_eval$accuracy * 100, 2), "%\n")
```

Accuracy: 86.21 %

```
cat("Precision:", round(valid_eval$precision, 4), "\n")
```

```
Precision: 0.7053
```

```
cat("Recall:", round(valid_eval$recall, 4), "\n")
```

```
Recall: 0.6256
```

```
cat("F1 Score:", round(valid_eval$f1, 4), "\n")
```

```
F1 Score: 0.6631
```

```
# Make predictions on test set
cat("\nMaking predictions on test set...\n")
```

```

Making predictions on test set...
```

```
test_preds <- predict_stacked(test_loan, Train_loan, stacked_model, "loan_status")
test_eval <- evaluate_classifier(test_preds$Actual, test_preds$Predicted)

cat("\nTest Set Performance:\n")
```

```

Test Set Performance:
```

```
cat("Confusion Matrix:\n")
```

```
Confusion Matrix:
```

```
print(test_eval$confusion_matrix)
```

```
           Actual
Predicted   defaulted paid
  defaulted       786  346
  paid            443 4178
```

```
cat("Accuracy:", round(test_eval$accuracy * 100, 2), "%\n")
```

```
Accuracy: 86.29 %
```

```
cat("Precision:", round(test_eval$precision, 4), "\n")
```

```
Precision: 0.6943
```

```
cat("Recall:", round(test_eval$recall, 4), "\n")
```

```
Recall: 0.6395
```

```
cat("F1 Score:", round(test_eval$f1, 4), "\n")
```

```
F1 Score: 0.6658
```

```r
# Compare with base models
# Load previous model results
# Re-evaluate the base models
#------------------------------------------------------------------------------
# Re-evaluate base models on test set
#------------------------------------------------------------------------------

# Prepare test data - ensure it's clean for consistent comparison
test_clean <- na.omit(test_loan)
test_clean$loan_status <- factor(test_clean$loan_status)

# Function to evaluate model
eval_model <- function(actual, predicted) {
  conf_matrix <- table(Predicted = predicted, Actual = actual)
  accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)

  # Calculate precision, recall, F1 for "defaulted" class if it exists in the data
  if ("defaulted" %in% unique(c(as.character(actual), as.character(predicted)))) {
    tp <- sum(predicted == "defaulted" & actual == "defaulted")
    fp <- sum(predicted == "defaulted" & actual != "defaulted")
    fn <- sum(predicted != "defaulted" & actual == "defaulted")

    precision <- ifelse(tp + fp > 0, tp / (tp + fp), 0)
    recall <- ifelse(tp + fn > 0, tp / (tp + fn), 0)
    f1 <- ifelse(precision + recall > 0, 2 * precision * recall / (precision + recall), 0)
  } else {
    precision <- NA
    recall <- NA
    f1 <- NA
  }

  return(list(
    accuracy = accuracy,
    precision = precision,
    recall = recall,
    f1 = f1
  ))
}

# Get all baseline model predictions for comparison
base_preds <- get_base_predictions(Train_loan, test_clean, "loan_status")

# Evaluate each base model
nb_eval <- eval_model(
  ifelse(base_preds$actual == 1, "defaulted", "paid"),
  ifelse(base_preds$nb_pred == 1, "defaulted", "paid")
)

lda_eval <- eval_model(
  ifelse(base_preds$actual == 1, "defaulted", "paid"),
  ifelse(base_preds$lda_pred == 1, "defaulted", "paid")
)
```

```r
log_eval <- eval_model(
  ifelse(base_preds$actual == 1, "defaulted", "paid"),
  ifelse(base_preds$log_pred == 1, "defaulted", "paid")
)

ridge_eval <- eval_model(
  ifelse(base_preds$actual == 1, "defaulted", "paid"),
  ifelse(base_preds$ridge_pred == 1, "defaulted", "paid")
)

tree_eval <- eval_model(
  ifelse(base_preds$actual == 1, "defaulted", "paid"),
  ifelse(base_preds$tree_pred == 1, "defaulted", "paid")
)

# Create comparison table
model_comparison <- data.frame(
  Model = c("Naive Bayes", "LDA", "Logistic Regression", "Ridge Regression", "Decision Tree", "Stacke
  Accuracy = c(
    nb_eval$accuracy,
    lda_eval$accuracy,
    log_eval$accuracy,
    ridge_eval$accuracy,
    tree_eval$accuracy,
    test_eval$accuracy
  ),
  F1_Score = c(
    nb_eval$f1,
    lda_eval$f1,
    log_eval$f1,
    ridge_eval$f1,
    tree_eval$f1,
    test_eval$f1
  )
)

# Sort by accuracy
model_comparison <- model_comparison[order(model_comparison$Accuracy, decreasing = TRUE), ]
model_comparison$Accuracy <- round(model_comparison$Accuracy * 100, 2)
model_comparison$F1_Score <- round(model_comparison$F1_Score, 4)

cat("\nModel Comparison:\n")
```

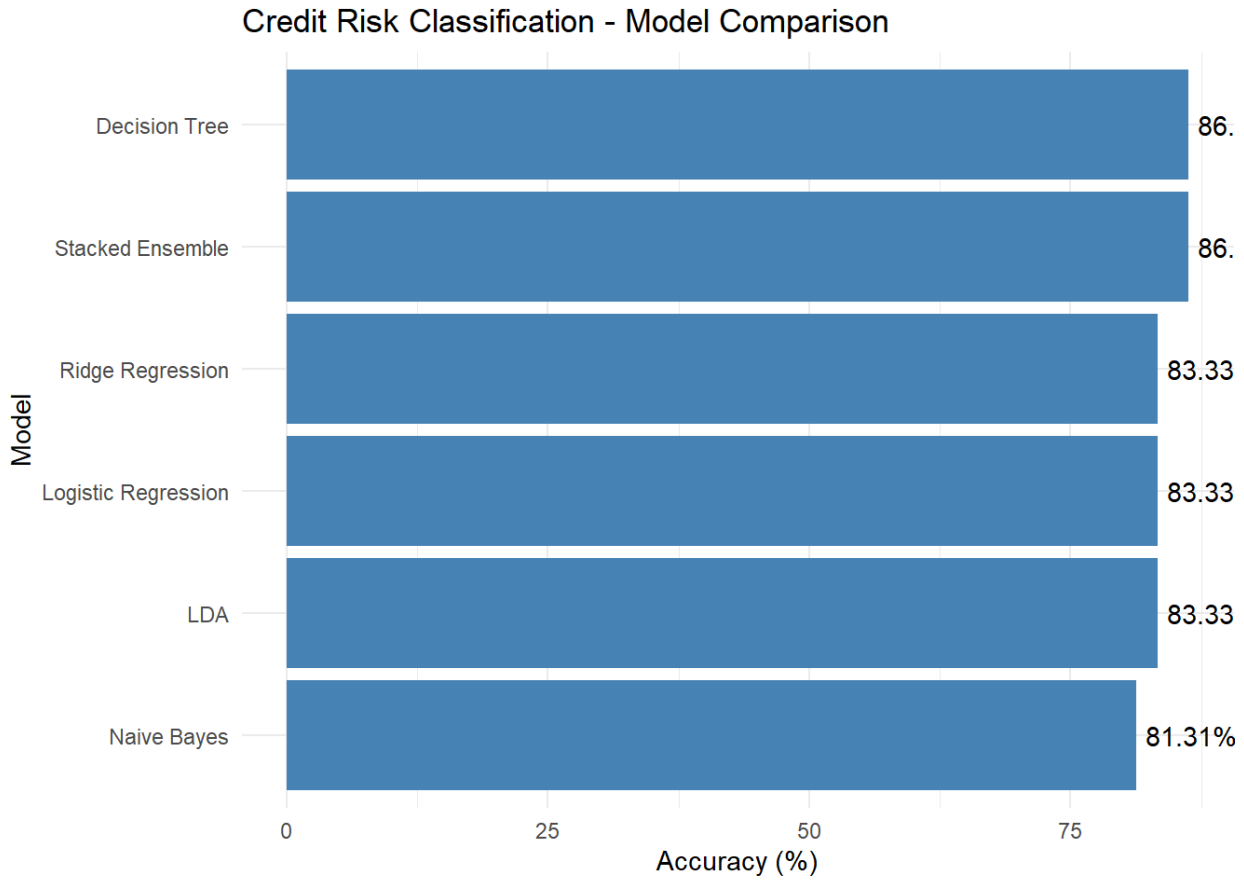
Model Comparison:

```r
print(model_comparison)
```

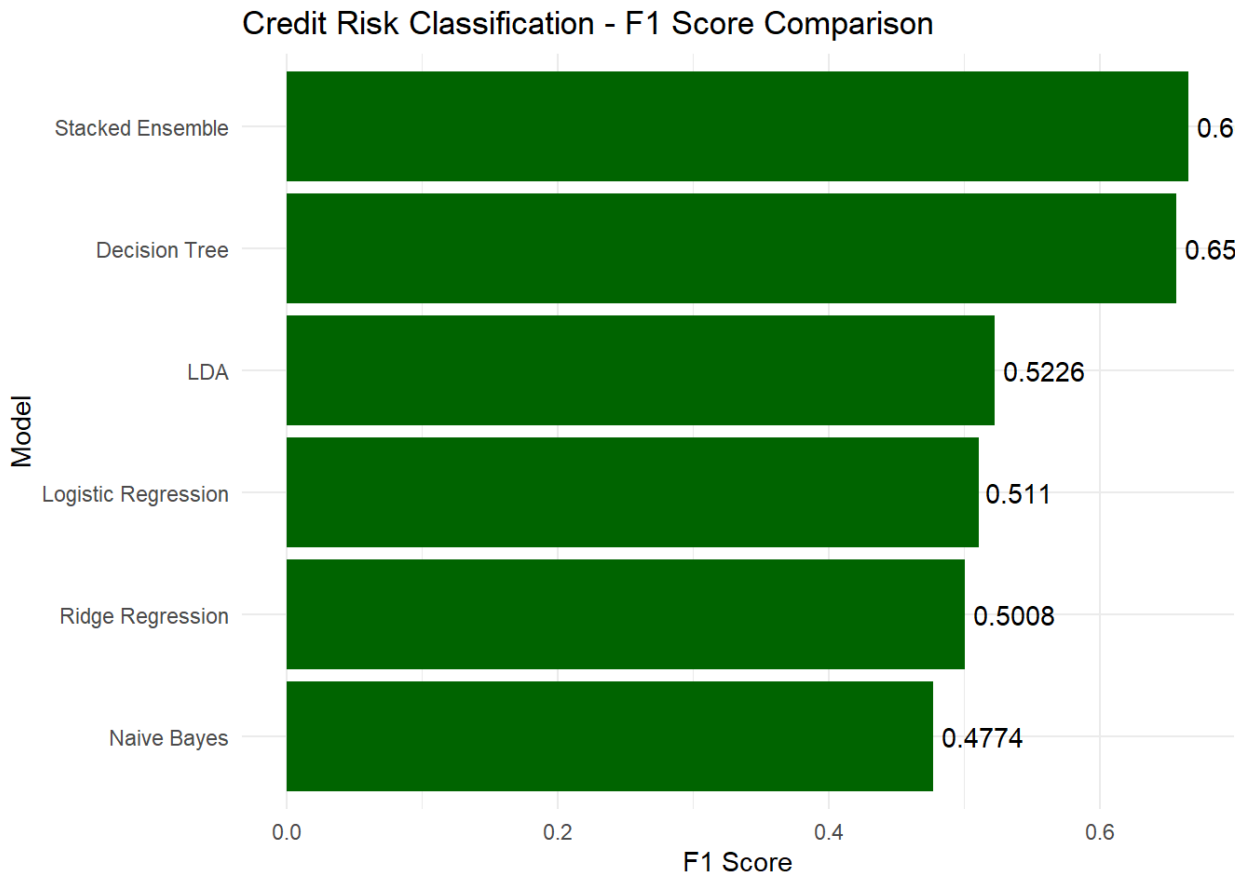
```
                Model Accuracy F1_Score
5       Decision Tree    86.32   0.6565
6    Stacked Ensemble    86.29   0.6658
2                 LDA    83.33   0.5226
```

```
3 Logistic Regression    83.33   0.5110
4    Ridge Regression    83.33   0.5008
1         Naive Bayes    81.31   0.4774
```

```r
# Plot model comparison
ggplot(model_comparison, aes(x = reorder(Model, Accuracy), y = Accuracy)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(x = "Model", y = "Accuracy (%)",
       title = "Credit Risk Classification - Model Comparison") +
  theme_minimal() +
  geom_text(aes(label = paste0(Accuracy, "%")), hjust = -0.1)
```




```r
# Plot F1 score comparison
ggplot(model_comparison, aes(x = reorder(Model, F1_Score), y = F1_Score)) +
  geom_bar(stat = "identity", fill = "darkgreen") +
  coord_flip() +
  labs(x = "Model", y = "F1 Score",
       title = "Credit Risk Classification - F1 Score Comparison") +
  theme_minimal() +
  geom_text(aes(label = F1_Score), hjust = -0.1)
```



The fairness constraint we employed was age, as naive bayes, LDA, logistic regression, ridge, and lasso likely discriminated based on age. This resulted in a lower fairness disparity, making decisions tree and stacked ensemble the fairest. In the stacked ensemble, age was a protected class, while in the decision tree, it examined features that would have the greatest impact on whether a person paid or defaulted on their loan. If a bank were to implement one of the models, it would implement a stacked ensemble with age as a protected class or a Decision Tree.

Data

https://www.kaggle.com/datasets/laotse/credit-risk-dataset

References

https://medium.com/@bneeraj026/logistic-regression-with-l2-regularization-from-scratch-1bbb078f1e88

https://www.brookings.edu/articles/fairness-in-machine-learning-regulation-or-standards/#defining-ml-fairness

https://www.deepchecks.com/glossary/model-fairness/#:~:text=It%20can%20arise%20from%20sources,compromising%20fairness%2C%20in%20machine%20learning.