AssignmentIsha

Isha M Borgaonkar 24209758

Exercise 2: Bat Species Classification Using Acoustic Features

Overview of What I exactly Performed in this code:

1) Loaded and Preprocessed the Data: I imported the dataset, checked its structure, and handled categorical variables.
2) Reduced Dimensions with PCA(Principle Component Analysis): To simplify the data, I applied PCA, keeping about 85% of the variance while reducing the number of features.
3) Trained and Evaluated Two Models: I trained a Multinomial Logistic Regression (C1) on the PCA-transformed data and a Neural Network (C2) on the original features.
4) Used 5-Fold Cross-Validation: I split the data into five folds to evaluate both models fairly and prevent overfitting.
5) Selected the Best Model: After comparing the accuracy of both models, I picked the one that performed better.
6) Made Predictions and Evaluated Performance: I used the best model to predict bat species and calculated overall accuracy, as well as accuracy specifically for the 'emba' family.

I have provided a detailed explanation for each block of code, describing its purpose and functionality in the report corresponding to each R chunk.

Step 1: Load and Explore the Data

```
# Load data_assignment_1_bats.RData dataset
load("data_assignment_1_bats.RData")

# Inspect structure of dataset
str(data_bats)
```

```
'data.frame':   7994 obs. of  73 variables:
 $ Family          : Factor w/ 4 levels "emba","morm",..: 3 4 1 4 4 3 4 4 3 4 ...
 $ CallDuration    : num  1.26 5.16 7.86 3.99 9.09 ...
 $ Fc              : num  49.2 36.6 44.3 14.4 27.2 ...
 $ HiFreq          : num  66.1 64.4 45.8 29.4 53.1 ...
 $ LowFreq         : num  47.2 31 41.2 13.9 25 ...
 $ Bndwdth         : num  18.88 31.26 3.53 15.08 27.96 ...
 $ FreqMaxPwr      : num  56.9 38.5 44.5 20.4 29.5 ...
 $ PrcntMaxAmpDur  : num  44.2 53.6 19 15.2 69.5 ...
 $ TimeFromMaxToFc : num  0.47 1.38 4.88 3.52 2.14 ...
 $ FreqKnee        : num  53 45.7 44.9 21.1 43.7 ...
 $ PrcntKneeDur    : num  27.2 21.6 79.7 31.5 25.9 ...
 $ StartF          : num  66.1 60.3 44.8 30 52.7 ...
 $ EndF            : num  47.2 30.9 41.3 13.9 24.9 ...
 $ DominantSlope   : num  10.5989 2.9468 0.0645 2.7326 3.3671 ...
 $ SlopeAtFc       : num  14.5107 3.9785 0.0637 2.9764 2.1363 ...
 $ StartSlope      : num  -16.434 -22.724 -0.253 -9.154 -12.274 ...
 $ EndSlope        : num  -14.76 -10.99 -3.46 -2.16 -1.67 ...
 $ SteepestSlope   : num  16.99 20.98 0.5 13.38 9.96 ...
 $ LowestSlope     : num  1.04e+01 2.18 5.41e-05 2.22 1.83 ...
 $ TotalSlope      : num  12.89 5.551 0.109 6.647 4.209 ...
 $ HiFtoKnSlope    : num  16.626 15.8801 0.0922 8.4186 4.858 ...
 $ KneeToFcSlope   : num  13.47 3.56 0.2 4.09 4.77 ...
 $ CummNmlzdSlp    : num  12.063 4.107 0.347 6.236 4.091 ...
 $ HiFtoFcExpAmp   : num  5.878 2.199 0.207 1.555 4.669 ...
 $ HiFtoFcDmp      : num  1.341 0.53 0.229 0.934 0.278 ...
 $ KnToFcExpAmp    : num  5.7154 2.403 0.0226 1.0783 3.7178 ...
 $ KnToFcDmp       : num  1.446 0.615 5.32 1.309 0.353 ...
 $ HiFtoKnExpAmp   : num  1.768 3.103 0.23 0.707 1.726 ...
 $ HiFtoKnDmp      : num  5.296 1.927 0.327 1.586 0.896 ...
 $ FreqLedge       : num  51 50.1 44.3 23.1 26 ...
 $ LedgeDuration   : num  0.638 1.96 3.663 0.202 2.331 ...
 $ FreqCtr         : num  49.8 43.2 44.3 21.1 36.4 ...
 $ FBak32dB        : num  108.7 58.4 45.9 27.8 42.5 ...
 $ FFwd32dB        : num  41.1 34.2 41.9 14.4 24.9 ...
 $ FBak20dB        : num  76.8 50.3 45.6 26.9 42.8 ...
 $ FFwd20dB        : num  49.5 36.3 42.6 15.9 28.4 ...
 $ FBak15dB        : num  76 48.7 45.3 26.6 41.3 ...
 $ FFwd15dB        : num  50.6 36.6 43.1 16.3 29.1 ...
 $ FBak5dB         : num  57 44 44.8 20.3 39.7 ...
 $ FFwd5dB         : num  49.4 38 43.8 17.9 30.1 ...
 $ Bndw32dB        : num  61.4 21 7.5 26.6 18.6 ...
 $ Bndw20dB        : num  18.67 15.27 2.91 8.38 14.05 ...
```

```
 $ Bndw15dB        : num  15.07 10.77 2.32 6.94 11.05 ...
 $ Bndw5dB         : num  8.25 3.94 1.08 4.38 3.98 ...
 $ DurOf32dB       : num  6.18 5 11.59 3.83 7.51 ...
 $ DurOf20dB       : num  1.04 3.39 8.7 2.96 6.46 ...
 $ DurOf15dB       : num  0.983 2.877 8.595 2.801 5.595 ...
 $ DurOf5dB        : num  0.592 1.369 4.679 1.101 1.365 ...
 $ Amp1stQrtl      : num  78.3 75.1 135 128.8 45.7 ...
 $ Amp2ndQrtl      : num  10865 15245 17774 8177 11668 ...
 $ Amp3rdQrtl      : num  21942 20412 14640 12866 14877 ...
 $ Amp4thQrtl      : num  21273 13862 12472 7972 9987 ...
 $ Amp1stMean      : num  0.67 0.765 0.924 0.892 0.684 ...
 $ Amp2ndMean      : num  0.892 0.936 0.972 0.845 0.922 ...
 $ Amp3rdMean      : num  0.961 0.983 0.939 0.917 0.935 ...
 $ Amp4thMean      : num  0.843 0.825 0.856 0.779 0.821 ...
 $ LnExpA_StartAmp : num  5.475 5.352 4.381 -1.894 -0.137 ...
 $ LnExpB_StartAmp : num  -2.19e-04 -2.61e-05 -8.48e-06 -3.61e-03 -1.85e-03 ...
 $ AmpStartLn60ExpC: num  5.69 5.6 4.93 4.08 4.09 ...
 $ LnExpA_EndAmp   : num  3.09 4.83 5.55 4.99 5.93 ...
 $ LnExpB_EndAmp   : num  -7.59e-04 -1.62e-05 -1.43e-04 -6.09e-05 -2.66e-05 -5.31e-05 -8.34e
 $ AmpEndLn60ExpC  : num  4.4 5.22 5.76 5.33 6.04 ...
 $ AmpK.start      : num  9.396 11.335 4.437 0.972 9.978 ...
 $ AmpK.end        : num  7.12 7.76 24.98 39.74 7.61 ...
 $ AmpKurtosis     : num  1.62 5.54 12.29 3.89 2.8 ...
 $ AmpSkew         : num  -0.273 -1.748 -2.633 -1.146 -0.663 ...
 $ AmpVariance     : num  0.0234 0.01433 0.0084 0.00935 0.03241 ...
 $ AmpMoment       : num  0.02327 0.01432 0.00826 0.00916 0.03222 ...
 $ AmpGausR2       : num  0.915 0.728 0.5 0.128 0.964 ...
 $ HiFminusStartF  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ FcMinusEndF     : num  2.6 7.71 2.12 3.57 1.82 ...
 $ RelPwr2ndTo1st  : num  0.354 0.303 0 0.74 0.557 ...
 $ RelPwr3rdTo1st  : num  0 0.0649 0.027 0.1951 0.0579 ...
 - attr(*, "na.action")= 'omit' Named int [1:414] 9 19 47 53 56 134 145 150 159 206 ...
  ..- attr(*, "names")= chr [1:414] "9" "19" "47" "53" ...
```

```r
# Check missing values in dataset
sum(is.na(data_bats))
```

```
[1] 0
```

```r
# Convert target variable to factor
data_bats$Family <- as.factor(data_bats$Family)
```

Description of step 1:

1)The code **loads and preprocesses** dataset `data_assignment_1_bats.RData`.

2)The `load()` function imports the dataset, `str()` function checks its structure as well as `sum(is.na())` identifies missing values.

3)This bats dataset has **73 variables** (72 numerical features, 1 categorical target).

4)The `as.factor()` function converts `Family` to a factor for classification.

5)This ensures data integrity, proper handling of categorical variables, and compatibility with machine learning models.

6)Pre processing prepares the dataset for **model training,feature selection and evaluation**, It optimizes predictive performance.

Step 2: Dimensionality Reduction using PCA

```
# Perform PCA
prcomp_result <- prcomp(data_bats[,-1], scale = TRUE)

# Compute cumulative variance explained
explained_variance <- cumsum(prcomp_result$sdev^2) / sum(prcomp_result$sdev^2)

# Choose Q components where cumulative variance  85%
Q <- which.min(abs(explained_variance - 0.85))

# Transform data using first Q components
data_pca <- as.data.frame(prcomp_result$x[,1:Q])
data_pca$Family <- data_bats$Family  # Add target variable back
```

Description of Step 2: Dimensionality Reduction using PCA

1)The **Principal Component Analysis (PCA)** applied on code to reduce dimensionality in `data_bats`.

2)The `prcomp()` function standardizes and computes **principal components**. It transfors the numerical features present in the dataset. `cumsum(prcomp_result$sdev^2) / sum(prcomp_result$sdev^2)` calculates the **cumulative variance explained** by components.

3)The `which.min(abs(explained_variance - 0.85))` selects the optimal **Q components**, retaining ~**85% variance**. The dataset is then get transformed by using the first **Q principal components** which is stored in `data_pca`.

4)Lastly, for classification tasks the `Family` column is get added.

5)Above code **reduces feature dimensionality**, reduction in feature dimentionality improves model efficiency, and minimizes overfitting while preserving most of the data variance.

Step 3: Train Classifiers and Perform Cross-Validation

```
set.seed(42)
n_folds <- 5  # Number of folds for cross-validation
folds <- sample(rep(1:n_folds, length.out = nrow(data_bats)))

accuracy_c1 <- c()
accuracy_c2 <- c()

for (i in 1:n_folds) {
  train_index <- folds != i
  test_index <- folds == i

  # Train Multinomial Logistic Regression (C1)
  model_c1 <- nnet::multinom(Family ~ ., data = data_pca[train_index, ])
  pred_c1 <- predict(model_c1, newdata = data_pca[test_index, ])
  acc_c1 <- mean(pred_c1 == data_pca$Family[test_index])
  accuracy_c1 <- c(accuracy_c1, acc_c1)

  # Train Neural Network (C2)
  model_c2 <- nnet::nnet(Family ~ ., data = data_bats[train_index, ], size = 5, maxit = 500)
  pred_c2 <- predict(model_c2, newdata = data_bats[test_index, ], type = "class")
  acc_c2 <- mean(pred_c2 == data_bats$Family[test_index])
  accuracy_c2 <- c(accuracy_c2, acc_c2)
}
```

```
# weights:  80 (57 variable)
initial  value 8865.352439
iter  10 value 3680.048918
iter  20 value 3572.718804
```

```
iter  30 value 3004.448926
iter  40 value 2611.233308
iter  50 value 2400.551706
iter  60 value 2198.290401
iter  70 value 1849.613908
iter  80 value 1842.494812
iter  90 value 1841.954378
iter 100 value 1841.933001
final  value 1841.933001
stopped after 100 iterations
# weights:  389
initial  value 10470.533151
iter  10 value 6942.918578
iter  10 value 6942.918576
iter  10 value 6942.918576
final  value 6942.918576
converged
# weights:  80 (57 variable)
initial  value 8865.352439
iter  10 value 3655.154518
iter  20 value 3520.522749
iter  30 value 3002.792846
iter  40 value 2624.554645
iter  50 value 2427.943987
iter  60 value 2214.014790
iter  70 value 1941.035898
iter  80 value 1935.272128
iter  90 value 1934.780793
iter 100 value 1934.641971
final  value 1934.641971
stopped after 100 iterations
# weights:  389
initial  value 7252.793620
iter  10 value 6880.721076
final  value 6880.715932
converged
# weights:  80 (57 variable)
initial  value 8865.352439
iter  10 value 3735.460675
iter  20 value 3602.714084
iter  30 value 2873.819376
iter  40 value 2675.850989
iter  50 value 2433.468470
```

```
iter  60 value 2217.710867
iter  70 value 1919.030892
iter  80 value 1908.239370
iter  90 value 1905.816097
iter 100 value 1905.579981
final  value 1905.579981
stopped after 100 iterations
# weights:  389
initial  value 11611.672034
iter  10 value 6965.346500
final  value 6915.325538
converged
# weights:  80 (57 variable)
initial  value 8865.352439
iter  10 value 3843.360798
iter  20 value 3702.711016
iter  30 value 3083.257385
iter  40 value 2639.064901
iter  50 value 2464.665767
iter  60 value 2266.164295
iter  70 value 1922.724188
iter  80 value 1914.044331
iter  90 value 1913.411666
iter 100 value 1913.407955
final  value 1913.407955
stopped after 100 iterations
# weights:  389
initial  value 10623.900484
iter  10 value 6933.138782
iter  20 value 6932.885316
iter  30 value 6931.638756
iter  40 value 6931.501204
final  value 6931.469010
converged
# weights:  80 (57 variable)
initial  value 8866.738734
iter  10 value 3674.940190
iter  20 value 3527.742456
iter  30 value 2892.134510
iter  40 value 2637.613710
iter  50 value 2412.122279
iter  60 value 2215.397286
iter  70 value 1954.955764
```

```
iter  80 value 1948.328929
iter  90 value 1947.938303
iter 100 value 1947.926961
final  value 1947.926961
stopped after 100 iterations
# weights:  389
initial  value 7270.513174
iter  10 value 6918.142840
iter  20 value 6860.004141
iter  30 value 6852.630998
iter  40 value 6499.486406
iter  50 value 6473.824134
iter  60 value 6465.777037
iter  70 value 6465.336432
iter  80 value 6437.291360
iter  90 value 6390.142573
iter 100 value 6388.104948
iter 110 value 6385.765140
iter 120 value 6385.131602
iter 130 value 6354.913658
iter 140 value 6348.063358
iter 150 value 6332.294160
iter 160 value 6274.121799
iter 170 value 6265.602985
iter 180 value 6128.654590
iter 190 value 5658.651684
iter 200 value 5210.017063
iter 210 value 4738.368250
iter 220 value 4524.278078
iter 230 value 4473.218616
iter 240 value 4426.087455
iter 250 value 4396.944954
iter 260 value 4355.567114
iter 270 value 4313.717824
iter 280 value 4299.774714
iter 290 value 4284.574930
iter 300 value 4274.864578
iter 310 value 4265.959000
iter 320 value 4261.315005
iter 330 value 4258.838625
iter 340 value 4255.961373
iter 350 value 4255.388364
iter 360 value 4253.565632
```

```
iter 370 value 4252.896619
iter 380 value 4251.928279
iter 390 value 4251.241916
iter 400 value 4249.553542
iter 410 value 4243.299159
iter 420 value 4241.316374
iter 430 value 4239.318613
iter 440 value 4238.480738
iter 450 value 4238.147696
iter 460 value 4236.308025
iter 470 value 4231.723925
iter 480 value 4228.529079
iter 490 value 4226.923109
iter 500 value 4224.805057
final  value 4224.805057
stopped after 500 iterations
```

Description of Step 3: Train Classifiers and Perform Cross-Validation

Above code implements **5-fold cross-validation** for evaluation of two classifiers: 1. **Multinomial Logistic Regression (C1) with PCA-transformed data** 2. **Neural Network (C2) with original numerical features**

Cross-Validation Setup

1)`set.seed(42)`: It ensures reproducibility of results.

2)`n_folds <- 5`: This syntax Specifies **5-fold** cross-validation.

3)`folds <- sample(rep(1:n_folds, length.out = nrow(data_bats)))`:Randomly assigns each observation to one of the 5 folds.

Training and Evaluation Process

The `for` loop iterates over the **5 folds**, and performs:

1. **Splitting Data**

- `train_index <- folds != i`: This identifies training data indices.
- `test_index <- folds == i`: This identifies test data indices.

2. **Multinomial Logistic Regression (C1) Training and Evaluation**

- `model_c1 <- nnet::multinom(Family ~ ., data = data_pca[train_index, ])`:
  - This code **trains C1** using **PCA-reduced features** (`data_pca`).
- `pred_c1 <- predict(model_c1, newdata = data_pca[test_index, ])`:
  - This code predicts class labels for test data.
- `acc_c1 <- mean(pred_c1 == data_pca$Family[test_index])`:
  - Computes **accuracy** as the proportion of correctly predicted labels.
- `accuracy_c1 <- c(accuracy_c1, acc_c1)`:
  - This syntax Stores accuracy for this fold(C1).

3. **Neural Network (C2) Training and Evaluation**

- `model_c2 <- nnet::nnet(Family ~ ., data = data_bats[train_index, ], size = 5, maxit = 500)`:
  - Above syntax trains **C2**, a neural network with **5 hidden units** and **500 iterations** on the **original numerical features** (`data_bats`).
- `pred_c2 <- predict(model_c2, newdata = data_bats[test_index, ], type = "class")`:
  - Predicts class labels for test data.
- `acc_c2 <- mean(pred_c2 == data_bats$Family[test_index])`:
  - Above line computes accuracy for **C2**.
- `accuracy_c2 <- c(accuracy_c2, acc_c2)`:
  - This syntax Stores accuracy for this fold(C2).

Key Impact on Data Processing

- **Feature Reduction for C1:** PCA reduces **high-dimensional features** to a smaller set. This improves computational efficiency.
- **Raw Feature Learning for C2:** The neural network utilizes **all 72 original features** and depends on its capability to identify and extract meaningful patterns.
- **Cross-Validation:** Ensures **robust model evaluation**, it prevents overfitting to a single dataset split.

Step 4: Select the Best Model and Evaluate Performance

```r
# Ensure that accuracy values exist
if (!exists("accuracy_c1") || !exists("accuracy_c2")) {
  stop("Error: Model accuracies not computed. Run the cross-validation step first.")
}

# Compute average accuracy for both models
mean_acc_c1 <- mean(accuracy_c1)
mean_acc_c2 <- mean(accuracy_c2)

# Select the best model
if (mean_acc_c1 > mean_acc_c2) {
  best_model <- model_c1
  best_model_name <- "Multinomial Logistic Regression + PCA"
} else {
  best_model <- model_c2
  best_model_name <- "Neural Network"
}

print(paste("Best Model Selected:", best_model_name))
```

```
[1] "Best Model Selected: Multinomial Logistic Regression + PCA"
```

Description of above code

1)Before calculating the average accuracy, I first checked if `accuracy_c1` and `accuracy_c2` exist.

2)Then, I compared the cross-validation accuracies of **Multinomial Logistic Regression with PCA (C1)** and the **Neural Network (C2)**.

3)The model with the best performance was assigned to `best_model`, and its name was stored in `best_model_name`.

```r
exists("best_model_name")
```

```
[1] TRUE
```

```r
# Ensure correct dataset for predictions
if (best_model_name == "Multinomial Logistic Regression + PCA") {
  predictions <- predict(best_model, newdata = data_pca, type = "class")
```

```
} else {
  predictions <- predict(best_model, newdata = data_bats, type = "class")

}

# Compute overall accuracy
overall_accuracy <- mean(predictions == data_bats$Family)
print(paste("Final Model Accuracy:", overall_accuracy))
```

```
[1] "Final Model Accuracy: 0.884788591443583"
```

```
# Compute accuracy for 'emba' family specifically
emba_accuracy <- mean(predictions[data_bats$Family == "emba"] == "emba")
print(paste("Accuracy for 'emba' Family:", emba_accuracy))
```

```
[1] "Accuracy for 'emba' Family: 0.883720930232558"
```

Description of step 4

1. **Choosing the Right Dataset for Predictions:**
   - First, the code checks which model performed best.
   - If the best model is **Multinomial Logistic Regression with PCA**, it uses the **PCA-transformed dataset** (`data_pca`) for predictions.
   - Otherwise, it sticks with the **original dataset** (`data_bats`).
2. **Making Predictions:**
   - Here, the `predict()` function is used to generate predictions based on the selected dataset.
   - The `type = "class"` syntax ensures that the predictions return class labels instead of probabilities.
3. **Calculating Overall Accuracy:**
   - The predicted labels are compared with the actual labels in `data_bats$Family`.
   - The percentage of correct predictions get stored in `overall_accuracy` and displayed.

4. **Checking Accuracy for the 'emba' Family:**
    - The code specifically checks the model predicted the **'emba'** family.
    - It filters only those cases where the actual label is `"emba"` and after that calculates the proportion of correct predictions.
    - This accuracy is stored in `emba_accuracy` and get printed.

Conclusion

1)Component Analysis helped simplify the dataset, making it more efficient while keeping most of the useful information.

2)Cross-validation ensured a fair **model evaluation**, reducing the risk of overfitting.

3)Both **models were compared**, and the best one was **chosen based on accuracy.**

4)Final predictions were made, and I checked how well the model classified all species, especially the 'emba' family.