Multiclass Classification of Bat Species Using ML

Multinomial (with PCA) vs Neural Network comparison

Jainel Liz Boban

Table of Contents

Load data and set input and output

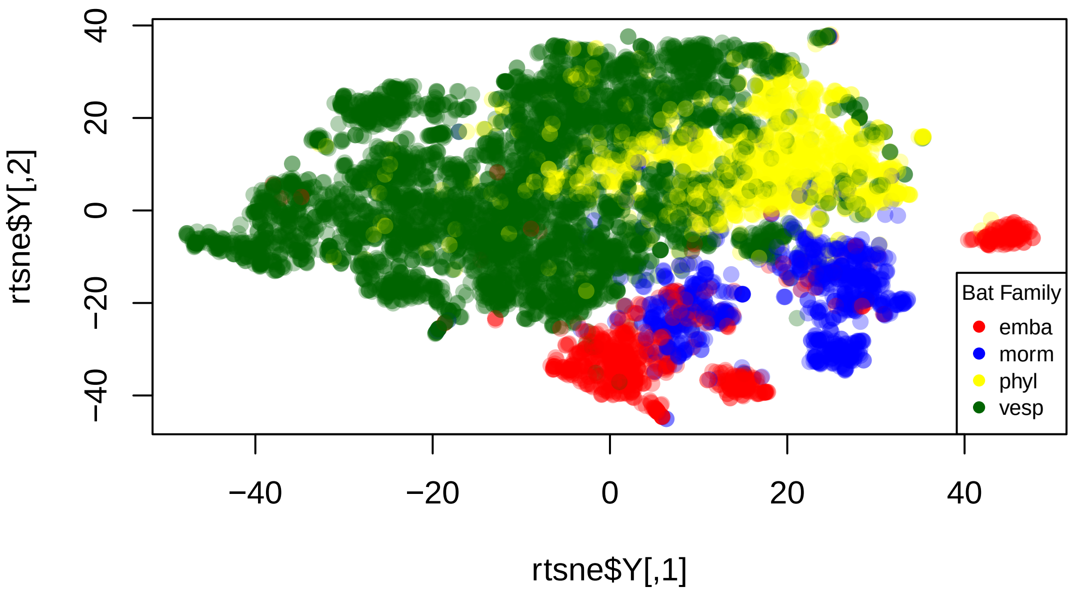
- Target variable **y** is extracted from 'Family' column of data-set.
- We will extract input feature matrix **x** from the data-set loaded by removing first column (Family).
- Input features are scaled to have 0 mean and standard deviation 1. This is done so that features with larger numerical values will not dominate the learning process.
- The scaling also helps in PCA feature selection and avoiding vanishing/exploding gradients.

```r
# load data frame data_bats using .RData
load("data_bats.RData")

# Extract 72 numerical input features (excluding 1st-Family column)
x <- scale(data_bats[, -1])  #standardize numerical features
# set y as target variable which is extracted from Family column
y=data_bats$Family
# Store total umber of observations (7994)
N <- nrow(x)
```

Visualize data

The t-Distributed Stochastic Neighbor Embedding (t-SNE) can be used for 2D visualization of high-dimensional data before model fitting.

```r
# Get four bat family labels for plotting purpose
family_labels <- levels(factor(y))
# Convert categorical labels to numeric with index starting at 0
y <- as.numeric(factor(y)) - 1

# use t-distributed stochastic neighbor embedding for visualizing in 2D
suppressWarnings(suppressMessages(library(Rtsne)))
# perplexity = nearest neighbors to consider
rtsne <- Rtsne(x, perplexity = 30)
# Apply 4 different colors for the classes
col_palette <- c("red", "blue", "yellow", "darkgreen")
cols <- col_palette[y+1]
# Plot using Rtnse results for 2D visualization
plot(rtsne$Y, pch = 19, col = adjustcolor(cols, 0.3))
legend("bottomright", legend = family_labels, col = col_palette, pch = 19,
          title = "Bat Family", cex=0.7)
```



rtsne$Y[,2]
40
20
0
−20
−40
−40
−20
0
20
40
rtsne$Y[,1]
Bat Family
● emba
● morm
● phyl
● vesp

Observations:

- The red (emba) and blue (morm) clusters appear well-separated from the rest (though there is some overlap between the two).
- There seems to be significant overlap between phyl (yellow) and vesp (green) classes, which could make classification challenging. In fact, Vesp is the largest and most scattered class.
- Overall, the data-set seem to contain distinguishing patterns for the bat families. A well-trained classifier should be able to classify them effectively.

Train-Test splitting of data

- We will split the data so that 25% of observations are available for testing (i.e. unseen data on which we will assess test performance).
- Remaining 75% fill be used to train and tune the model . (This portion of data will further be split into train and validation during cross-validation method).

```r
# set seed for reproducibility
set.seed(123)
# splitting of data
M <- round(0.25*N)  # around 25% of observations put in test set
test_index <- sample(1:N,M)  # randomly sample test set indices
x_test <- x[test_index,]  # test input set
x_train <- x[-test_index,]  # train input set
y_test <- y[test_index] # target test values
y_train <- y[-test_index] # target values for training set
```

Cross-Validation Method used- K fold

- We will use -fold validation to compare (using validation accuracy) and tune the two classifiers (to get hyper-parameter Q and H in each case).
- In this method, the data is divided into K roughly equally sized folds (we will use k=5 equal parts).
- On each iteration a fold is dropped and the model is fit using remaining k-1 folds.The predictive performance is evaluated on dropped fold.
- K-fold has been selected over leave-one-out method as it is computationally expense than k-fold. Also, a simple hold-out method might learn patterns that do not generalize well, leading to poor performance on test/unseen data.

```r
# number of folds in K-fold method
K <- 5
# Randomly sample and assign data to K=5 folds
k_folds <- sample(rep(1:K, length.out = nrow(x_train)))
```

C1 Classifier -Multinomial+PCA

Perform PCA on entire training data-set for principal components

- The data-set is high-dimensioal and has 72 input features. Hence performing PCA would be help in dimension reduction and therefore reduce complexity and avoid over-fitting.
- Instead of using all 72 features, we will use principal components that explain 80-90% variance as shown in below code.
- *prcomp()* is used to perform PCA on input features. Proportion of variance explained by Q principal components is given by cumulative variance explain by Q components by total variance explained by all components.

```r
# Perform PCA on training set
pca_model <- prcomp(x_train)
# Compute cumulative variance to determine range of Q
prop <- cumsum(pca_model$sdev^2) / sum(pca_model$sdev^2)
# Q values for which variance range is approx. 80-90%
Q_min <- min(which(prop >= 0.80))  # first Q after 80% variance
Q_max <- max(which(prop <= 0.90))  # last Q before 90% variance
Q_range <- Q_min:Q_max  # Range of Q values (80-90% variance)
Q_range
```

```
 [1] 14 15 16 17 18 19 20 21 22 23
```

- We can see that the variance explained by 14-23 first principal components explain 80-90% variance.
- We will use k-fold method to tune and find best Q from these 10 values.

Cross-Validation (K-fold) to find best Q

- We will use *x_train* within the k-fold loop for training and validation. So it is split into *k_fold_x_train* and *k_fold_x_val*.
- The k-fold method splits the *k_fold_x_train* data-set into 5 groups. In each iteration validation is performed on the dropped fold. So this step itself splits the data-set to validation in each step.
- On top of this we have a separate validation set *k_fold_x_val* for the purpose of comparing different **Q** (number of principal components).
- An additional loop iterates over the 10 values of Q that explains 80-90% variance in data.
- Accuracy and loss of both training and validation sets from each k-fold iteration have been stored in separate matrices.
- *nnet* library has been used to fit multinomial model for each k-fold and Q value using *multinom()* function.

```r
# Load necessary libraries for multinomial logistic regression
suppressWarnings(suppressMessages(library(nnet)))

# Initialize matrices to store accuracy and loss results
acc_train <- acc_val <- loss_train <- loss_val <- matrix(NA, K, length(Q_range))

# Loop over each fold (K-fold Cross-validation)
for (k in 1:K){
  # Split data into train and validation folds
  k_fold_val_index <- which(k_folds == k)
  k_fold_train_index <- setdiff(seq_len(nrow(x_train)), k_fold_val_index)
  k_fold_x_train <- x_train[k_fold_train_index,]
  k_fold_x_val <- x_train[k_fold_val_index,]
  k_fold_y_train <- y_train[k_fold_train_index]
  k_fold_y_val <- y_train[k_fold_val_index]

  for (q in seq_along(Q_range)){  # Iterate over Q values
    Q <- Q_range[q]
    # Perform PCA model on train and validation sets before fitting model
    k_x_train_pca <- predict(pca_model, newdata = k_fold_x_train)[, 1:Q]
    k_x_val_pca <- predict(pca_model, newdata = k_fold_x_val)[, 1:Q]
    # Fit multinomial regression using new PCA-based variables
    k_fold_multinom_fit <- multinom(k_fold_y_train ~ .,
           data = data.frame(k_fold_y_train, k_x_train_pca), trace = FALSE)
    # Predictions on training set
    y_train_predict <- predict(k_fold_multinom_fit,
                               newdata = data.frame(k_x_train_pca))
    # Training accuracy
    acc_train[k, q] <- mean(y_train_predict == k_fold_y_train)
    # Predictions on validation set
    y_val_predict <- predict(k_fold_multinom_fit,
                             newdata = data.frame(k_x_val_pca))
    # validation accuracy
    acc_val[k, q] <- mean(y_val_predict == k_fold_y_val)
    # loss on training set (categorical cross entropy)
    y_train_predict_prob <- predict(k_fold_multinom_fit,
              newdata = data.frame(k_x_train_pca), type = "probs")
    y_train_predict_prob <- pmax(y_train_predict_prob, 1e-20)  # Avoid log0 =-Inf
    loss_train[k, q] <--mean(log(y_train_predict_prob[cbind(seq_along(k_fold_y_train),
                               k_fold_y_train+1)]))
    # loss on validation set
    y_val_predict_prob <- predict(k_fold_multinom_fit,
                                  newdata = data.frame(k_x_val_pca),
                                   type = "probs")
    y_val_predict_prob <- pmax(y_val_predict_prob, 1e-20)
    loss_val[k, q] <- -mean(log(y_val_predict_prob[cbind(seq_along(k_fold_y_val),
                           k_fold_y_val+1)]))
  }
}
```

Find best Q with highest mean accuracy on validation set

- Best **Q** is selected to be one with highest validation accuracy overall all 5 folds.
- This is done by taking mean of all validation accuracies stored in matrix acc_val over all k folds for each **Q**.

```
# sverage accuracy and loss for each Q across folds
mean_acc_train <- tapply(acc_train, rep(Q_range, each = K), mean, na.rm = TRUE)
mean_acc_val <- tapply(acc_val, rep(Q_range, each = K), mean, na.rm = TRUE)
mean_loss_train <- tapply(loss_train, rep(Q_range, each = K), mean, na.rm = TRUE)
mean_loss_val <- tapply(loss_val, rep(Q_range, each = K), mean, na.rm = TRUE)
# Find best Q based on highest mean validation accuracy
best_Q <- as.numeric(names(mean_acc_val))[which.max(mean_acc_val)]

# Print results
cat("Best Q based on highest mean accuracy (on validation) =", best_Q, "\n")
```

```
Best Q based on highest mean accuracy (on validation) = 22
```

```
cat("Variance explained by best Q (",best_Q,") =", round(prop[best_Q] * 100,4),
    "%\n\n")
```

```
Variance explained by best Q ( 22 ) = 88.6417 %
```

```
cat("Average validation accuracy (highest with best Q) =",
             round(mean_acc_val[which.max(mean_acc_val)], 4), "\n")
```

```
Average validation accuracy (highest with best Q) = 0.8828
```

```
cat("Average training accuracy for best Q =",
             round(mean_acc_train[which.max(mean_acc_val)],4), "\n")
```

```
Average training accuracy for best Q = 0.8888
```

```
cat("Average validation loss corresponding to best Q =",
             round(mean_loss_val[which.max(mean_acc_val)],4),"\n")
```

```
Average validation loss corresponding to best Q = 0.3063
```

```
cat("Average training loss corresponding to best Q =",
             round(mean_loss_train[which.max(mean_acc_val)],4),"\n")
```

```
Average training loss corresponding to best Q = 0.2834
```

Perform PCA once again and fit model using best best Q

- PCA has been applied on x_train ($k\_fold\_x\_train + k\_fold\_x\_val$).
- Multinomial regression model has been fitted once again with the best Q , i.e., 22 principal components.

```
# Train model with best Q using entire training data-set
x_train_pca <- predict(pca_model,newdata = x_train)[,1:best_Q]
multinomial_fit <- multinom(y_train ~ ., data = data.frame(y_train, x_train_pca),
                             trace = FALSE)
```

Test data performance

- PCA has been performed on x_test which is unseen by both PCA and multinomial model.
- Target variable has been predicted using this new pca applied data together with multinomial fit.
- Accuracy for test data is given by $\frac{1}{M}\sum_{i=1}^{M}\mathbb{1}(\bar{y}_i^* = y_i^*)$ i.e. mean of number of correctly classified instances.

- For loss, we will need probabilities to calculate cross-entropy of categorical variables.

```
# Perform PCA on test set and predict based on new principal component features
x_test_pca <- predict(pca_model, newdata = x_test)[, 1:best_Q]
y_test_predict <- predict(multinomial_fit, newdata = data.frame(x_test_pca))
# accuracy of prediction on test set
acc_test <- mean(y_test_predict == y_test)

# get prediction probabilities instead of direct class values on validation set
y_test_predict_prob <- predict(multinomial_fit, newdata = data.frame(x_test_pca),
                      type = "probs")
y_test_predict_prob <- pmax(y_test_predict_prob, 1e-20)
# test loss using cross-entropy using model with best Q
loss_test <- -mean(log(y_test_predict_prob[cbind(seq_along(y_test),y_test+1)]))

# print confusion matrix
confusion_matrix <- table(Predicted = y_test_predict, Actual = y_test)
print(confusion_matrix)
```

```
         Actual
Predicted    0    1    2    3
        0  174   23    0    3
        1   21  181    4    6
        2    1    4  231   56
        3    5    0  115 1174
```

```
cat("\n")
```

```
cat("Final model test accuracy with best Q (",best_Q, ") is ",
                     round(acc_test,4),"\n")
```

```
Final model test accuracy with best Q ( 22 ) is  0.8809
```

```
cat("Final model test loss with best Q (",best_Q, ") is ",
                     round(loss_test,4),"\n")
```

```
Final model test loss with best Q ( 22 ) is  0.3109
```

C2 Classifier -Neural network

Train-test splitting

- The same *x_train* and *x_test* has been used in classifier C2.. However, y needs to be converted to one-hot encoding that will have 4 digits to represent the 4 classes.
- *y_train_one_hot* and *y_test_one_hot* need to extracted again from the *y_one_hot* which has 4 columns now.
- *keras3* library has been to fit neural network *to_categorical()* function converts **y** to one hot encoding.

```
# Load keras library for nueral network
suppressWarnings(suppressMessages(library(keras3)))

# Perform one-hot encoding on target variable for neural network
num_classes <- length(unique(y))  # number of unique classes = 4
y_one_hot <- keras::to_categorical(y, num_classes = num_classes)
```

```
Registered S3 methods overwritten by 'keras':
  method                               from
  as.data.frame.keras_training_history keras3
  plot.keras_training_history          keras3
  print.keras_training_history         keras3
  r_to_py.R6ClassGenerator             keras3
```

```
y_test_one_hot <- y_one_hot[test_index, ]
y_train_one_hot <- y_one_hot[-test_index, ]
```

Table 1: One-Hot Encoding Mapping

y	class.label	y_one_hot
0	emba	1000
1	morm	0100
2	phyl	0010
3	vesp	0001

Cross-Validation (K-fold) to find best H

- In classifier C2, similar cross-validation process has been followed as C1 classifier. An inner loop has been used within each k-fold (5-fold) iteration to tune **H**, i.e., number of hidden units. The number of hidden units have been passed as a list of selected numbers.
- For each **H** value within each fold, the neural network model is fitted. The accuracy and validation for each value of **H** are being stored in separate matrices.
- The hidden layer activation function is set to "**ReLU**"(Rectified Linear Unit). This is because it is less prone to vanishing gradient problem (derivative always 1 for positive inputs) compared to activation functions such as sigmoid and hyperbolic tangent.
- The output activation function is **"Softmax"** as the target variable comprises of multiclass discrete output values (for 4 bat classes). It was also chosen based on the error function used which is categorical cross-entropy. Modelling on discrete data is a hard task and Softmax function provides class probabilities instead of directly giving scores of each class.
- There is only **one hidden layer** and **an output layer** in this neural network model.

```r
V <- ncol(x) # number of columns of x

# Define the number of neurons to tune
H_vec <- c(0, 5, 10, 15, 20, 25)
H <- length(H_vec)
# matrices to store train and validation accuracy based on H
acc_train_H <- acc_val_H <- loss_train_H <- loss_val_H <- matrix(NA, K, H)

for (k in 1:K){# Loop over k folds
  # Split data into train and validation folds from x_train
  k_fold_val_index <- which(k_folds == k)
  k_fold_train_index <- setdiff(seq_len(nrow(x_train)), k_fold_val_index)
  k_fold_x_train <- x_train[k_fold_train_index,]
  k_fold_x_val <- x_train[k_fold_val_index,]
  k_fold_y_train <- y_train_one_hot[k_fold_train_index,]
  k_fold_y_val <- y_train_one_hot[k_fold_val_index,]

  #  loop over H index values to get optimal number of hidden units
  for (h in 1:H) {
    # Define model with different hidden layer sizes in Hvec
    k_model <- keras_model_sequential()
    k_model %>%
      # hidden layer activation function set to RELU
      layer_dense(units = H_vec[h], activation = "relu", input_shape = V) %>%
      # output layer activation function set to softmax
      layer_dense(units = num_classes, activation = "softmax")
    # Compile the model
    k_model %>% compile(
      loss = "categorical_crossentropy", # for multinomial classification
      optimizer = optimizer_sgd(), # stochasic gradient descent
      metrics = "accuracy"
      )
    # Train the model
    fit <- k_model %>% fit(
      x = k_fold_x_train, y = k_fold_y_train,
      validation_data = list(k_fold_x_val, k_fold_y_val),
      epochs =50,batch_size =32,
      verbose = 0  # remove training output
    )
    n_epoch <- fit$params$epochs  # Get number of epochs
    #Find accuracy of train and validation set using last epoch
    acc_train_H[k, h] <- fit$metrics$accuracy[n_epoch]
    acc_val_H[k, h] <- fit$metrics$val_accuracy[n_epoch]
    loss_train_H[k, h] <- fit$metrics$loss[n_epoch]
    loss_val_H[k, h] <- fit$metrics$val_loss[n_epoch]
  }
}
```

Find best H with highest mean accuracy on validation set

- Best **H** is selected to be one with highest validation accuracy overall all 5 folds.
- This is done by taking mean of all validation accuracies stored in matrix *acc_val_H* over all k folds for each **H**.

```
# Average accuracy and loss for each H across folds
mean_acc_train_H <- tapply(acc_train_H, rep(H_vec, each = K), mean, na.rm = TRUE)
mean_acc_val_H <- tapply(acc_val_H, rep(H_vec, each = K), mean, na.rm = TRUE)
mean_loss_train_H <- tapply(loss_train_H, rep(H_vec, each = K), mean, na.rm = TRUE)
mean_loss_val_H <- tapply(loss_val_H, rep(H_vec, each = K), mean, na.rm = TRUE)
# Find best H based on highest validation accuracy
H_best <- as.numeric(names(mean_acc_val_H))[which.max(mean_acc_val_H)]

# Print results
cat("Best number of neurons (H) based on highest mean validation accuracy =",
          H_best, "\n")
```

```
Best number of neurons (H) based on highest mean validation accuracy = 15
```

```
cat("Highest average validation accuracy (with best H) =",
              round(mean_acc_val_H[which.max(mean_acc_val_H)],4), "\n")
```

```
Highest average validation accuracy (with best H) = 0.9353
```

```
cat("Training accuracy for best H =",
              round(mean_acc_train_H[which.max(mean_acc_val_H)], 4), "\n")
```

```
Training accuracy for best H = 0.9461
```

```
cat("Validation loss corresponding to best H =",
              round(mean_loss_val_H[which.max(mean_acc_val_H)],4), "\n")
```

```
Validation loss corresponding to best H = 0.1978
```

```
cat("Training loss corresponding to best H =",
              round(mean_loss_train_H[which.max(mean_acc_val_H)],4), "\n")
```

```
Training loss corresponding to best H = 0.1576
```

Fit the final model using best H and get accuracy and loss on overall training set

- The neural network model has been fitted on *x_train* (*k_fold_x_train* + *k_fold_x_val*) with the best H , i.e., 15 units.

```
# Train final model with best H on train and validation sets
neural_model<-keras_model_sequential()
neural_model%>%
  layer_dense(units =H_best,activation ="relu",input_shape =V)%>%
  layer_dense(units =num_classes,activation ="softmax")
neural_model%>%
  compile(
    loss ="categorical_crossentropy",
    optimizer =optimizer_sgd(),
    metrics ="accuracy"
    )
neural_fit<-neural_model%>%fit(
  x =x_train,y =y_train_one_hot,
  epochs =50,batch_size =32,
  verbose=0
  )
neural_fit
```

```
Final epoch (plot to see history):
    loss: 0.1475
accuracy: 0.95
```

Selected Classifer = C2

- Classifier C2 consisting of neural network with 15 units has higher validation accuracy (93.53%) compared to Classier C1 consisting of multinomial logistic regression with PCA of 22 principal components (88.28%). This means C1 has better generalization compared to C2.
- It can be also seen that C2's validation loss is lower (0.2) compared to C1 (0.31). This could mean C2 has better convergence.
- Test results also show similar trend with C2 being better. C1 classifier shows a test accuracy of 88.09% and a test loss of 31.09%. It will be seen in next steps that C2 has a test accuracy of 94.39% and test loss of 16.77%.
- Both classifiers used the same cross-validation method (K-fold with k=5).
- Overall, as C2 has better performance metrics, model C2 can be selected among the two classifiers for next steps.
- This higher performance of C2 could be due to the fact that the neural network classifier C2 learns from all original 72 input features, whereas PCA-based classifier C1 lose some information as it reduces feature dimension to 22.
- Classifier C1 assumes a linear decision boundary but our dataset may require more complex boundaries.
- The neural network uses multiple hidden units (15) and uses non-linear activation functions such as ReLU and Softmax to capture complex patterns between input and target.

Classifier C2- Test performance

- The final model with best H is used to predict target variable on unseen data *x_test*.
- Accuracy and loss are directly obtained from the evaluation.

```
# evaluate final neural model on test (unseen) data and get test accuracy
y_test_evaluate <- neural_model %>% evaluate(x_test, y_test_one_hot, verbose=0)
test_accuracy <- y_test_evaluate[["accuracy"]]
test_loss     <- y_test_evaluate[["loss"]]
cat("Final test accuracy of classifier C2:", round(test_accuracy, 4), "\n")
```

```
Final test accuracy of classifier C2: 0.9439
```

```
cat("Final test loss of classifier C2:", round(test_loss, 4), "\n")
```

```
Final test loss of classifier C2: 0.1677
```

Observations:

- The neural network classifier C2 correctly predicts 94.39% of unseen samples.
- This test accuracy is also close to validation accuracy of 93.53%.
- Therefore, it can be said that the model generalizes well to unseen data.

Class 'emba' prediction ability

- Using the neural network model, the predicted class probabilities has been used to extract the most probable class as the predicted label.
- The predicted and actual indices are converted into class names to form a confusion matrix.

- The accuracy, precision and recall for *emba* class is obtained directly from the confusion matrix as per formula:

$$\text{Accuracy}_{\text{Emba}} = \frac{TP_{\text{Emba}} + TN_{\text{Emba}}}{\text{Total test samples}}$$

$$\text{Precision}_{\text{Emba}} = \frac{TP_{\text{Emba}}}{TP_{\text{Emba}} + FP_{\text{Emba}}}$$

$$\text{Recall}_{\text{Emba}} = \frac{TP_{\text{Emba}}}{TP_{\text{Emba}} + FN_{\text{Emba}}}$$

- where:
  - TP_{Emba} = True Positives (correctly classified as Emba);
  - TN_{Emba} = True Negatives (correctly classified not Emba);
  - FP_{Emba} = False Positives (other classes wrongly classified as Emba);
  - FN_{Emba} = False Negatives (true Emba class misclassified as another class).

```r
# Predicted target probabilities using final neural model
y_test_predict_prob <-neural_model %>% predict(x_test, verbose = 0)
# get index of class with highest probability
pred_class_index <- apply(y_test_predict_prob, 1, which.max) - 1
# convert probabilities to class labels
pred_class_labels <- factor(pred_class_index, levels = c(0, 1, 2, 3),
                           labels = c("emba", "morm", "phyl", "vesp"))
# Get actual labels from one-hot encoding
actual_class_index <- apply(y_test_one_hot, 1, which.max) - 1
actual_class_labels <- factor(actual_class_index, levels = c(0, 1, 2, 3),
                           labels = c("emba", "morm", "phyl", "vesp"))
# Form confusion matrix
confusion_matrix <- table(Predicted = pred_class_labels,
                          Actual = actual_class_labels)
print(confusion_matrix)
```

```
         Actual
Predicted emba morm phyl vesp
     emba  184    6    0    2
     morm   13  199    3    4
     phyl    0    3  295   25
     vesp    4    0   52 1208
```

```r
# get classification accuracy for emba class from confusion matrix
emba_true_positives <- confusion_matrix["emba","emba"] # true positives
emba_true_negatives <- sum(confusion_matrix[-which(rownames(confusion_matrix) == "emba"),
                    -which(colnames(confusion_matrix) == "emba")]) # true negatives
emba_accuracy <-(emba_true_positives + emba_true_negatives)/sum(confusion_matrix)

#Get precision and recall for emba class from confusion matrix
emba_false_positives <- sum(confusion_matrix["emba",]) - emba_true_positives
emba_false_negatives <- sum(confusion_matrix[,"emba"]) - emba_true_positives
emba_precision <- emba_true_positives / (emba_true_positives + emba_false_positives)
emba_recall <- emba_true_positives / (emba_true_positives + emba_false_negatives)

cat("\nAccuracy of classifier C2 for predicting the ‘Emba’ class:", round(emba_accuracy*100,4)
    ,"%\n")
```

```

Accuracy of classifier C2 for predicting the ‘Emba’ class: 98.7487 %
```

```r
cat("Precision of classifier C2 for predicting the ‘Emba’ class:",round(emba_precision*100,4),
    "%\n")
```

```
Precision of classifier C2 for predicting the ‘Emba’ class: 95.8333 %
```

```r
cat("Recall of classifier C2 for predicting the ‘Emba’ class:",round(emba_recall*100,4),
    "%\n")
```

```
Recall of classifier C2 for predicting the ‘Emba’ class: 91.5423 %
```

Observations:

- 98.75% accuracy indicates that classifier C2 has overall good predictive performance and correctly identifies most ‘Emba’ classes for the given test samples.
- 95.83% precision shows that the classifier is reliable as many of the predicted ‘Emba’ cases were correct.
- 91.54% recall shows classifier’s ability to detect many true ‘Emba’ labels correctly. From the confusion matrix, we can see that there are a few misclassifications mainly as ‘Morm’(13). This was expected from the t-SNE visualization which showed some overlap with ‘Emba’ class.