Kaggle Challenge 4

Diksha Shrestha

2023-04-25

Introduction

As for the Kaggle Challenge 4, we had an opportunity to work on improving any of the previous Kaggle Challenge so I decided to work with the first Kaggle Challenge which is the recommending species which are most likely to be observed by implementing the Autoencoder.

The Kaggle Challenge 3 helped me in understanding the concept of neural network better so I was able to join dots for implementing the Autoencoder for the eBird dataset.

Importing the dataset

I started with importing my training and test set csv file through the `read.csv()` function which was then converted to a matrix format using the `as.matrix()` function.

```
# importing the data
training_set <- as.matrix(read.csv("training_set.csv", row.names = 1))
test_set <- as.matrix(read.csv("test_set.csv", row.names = 1))
```

Implementation of Autoencoder

Autoencoder can be used as a component which helps in encoding the input data into a lower dimensional representation which is then used to compute the similarity measures between different species. Also, by comparing the encoded representations, the model can recommend similar species based on their observed patterns. So, I have tried implementing the autoencoder below:

The autoencoder is then trained using a forward-backward pass by the function. It runs through the columns (features) of the input data for each of the specified number of epochs. The element-wise maximum of the dot product of the input data (x) and weights (w) is used by the encoder to determine the hidden layer's activation, which is then followed by a relu activation. Similarly, the decoder takes the element-wise maximum of the dot product of the hidden layer activation (a1) and the weights' transposition (w), then performs a ReLU activation to determine the activation of the output layer. Furthermore, the code below also implements the Adam optimizer to update the weights.

```
autoencoder <- function(x, k = 128, learning_rate = 0.001, epochs = 20, beta1 = 0.9, beta2 = 0.999,tol =
  set.seed(123)
  w <- matrix(abs(rnorm(nrow(x) * k, sd = sqrt(2 / nrow(x)))), nrow(x), k)
  err <- rep(0, epochs)
  m <- v <- t <- 0
```

```
  total_error <- 0
  for (epoch in 1:epochs) {

    for (i in 1:ncol(x)) {
      # encoder
      a1 <- pmax(x[, i] %*% w, 0) # relu
      a2 <- pmax(a1 %*% t(w), 0) # relu
      # calculating error
      error <- x[, i] - a2
      err[epoch] <- err[epoch] + mean(abs(error))
      total_error <- total_error + sum(abs(error))
      # decoder
      a2_delta <- error * (a2 > 0)
      a1_delta <- a2_delta %*% w * (a1 > 0)

      dW <- x[, i] %*% a1_delta + t(t(a1) %*% a2_delta)
      #Adam Optimization
      t <- t + 1
      m <- beta1 * m + (1 - beta1) * dW
      v <- beta2 * v + (1 - beta2) * dW^2
      m_hat <- m / (1 - beta1^t)
      v_hat <- v / (1 - beta2^t)

      w <- w + learning_rate * m_hat / (sqrt(v_hat) + 1e-8)

      # apply non-negativity constraints
      w <- pmax(w, 0)

    }
    err[epoch] <- err[epoch] / ncol(x)
    cat(paste0("epoch: ", epoch,  "\n"))
    tol_ <- 1
    if (epoch > 1) {
      tol_ <- abs(err[epoch - 1] - err[epoch]) / (err[epoch] + err[epoch - 1])
    }
    cat("epoch: ", epoch, ", error: ", err[epoch], ", tol: ", tol_, "\n")
    if (tol_ < tol) break
  }
  list(w = as.matrix(w), h = as.matrix(t(m) %*% x))
}
```

Here, the autoencoder function is called which is then passed on the training_set as the input data. It is executed with the parameters such as hidden units, learning rates, epochs, and optimization parameter.

```
# Train the autoencoder model
autoencoder_model <- autoencoder(training_set)
```

```
## epoch: 1
## epoch:  1 , error:  6.852759 , tol:  1
## epoch: 2
## epoch:  2 , error:  0.2801629 , tol:  0.9214451
## epoch: 3
## epoch:  3 , error:  0.1251922 , tol:  0.3823083
```

```
## epoch: 4
## epoch:  4 , error:  0.07541298 , tol:  0.2481454
## epoch: 5
## epoch:  5 , error:  0.056607 , tol:  0.142448
## epoch: 6
## epoch:  6 , error:  0.04397801 , tol:  0.1255554
## epoch: 7
## epoch:  7 , error:  0.03404409 , tol:  0.1273219
## epoch: 8
## epoch:  8 , error:  0.03297786 , tol:  0.01590868
## epoch: 9
## epoch:  9 , error:  0.03197279 , tol:  0.01547429
## epoch: 10
## epoch:  10 , error:  0.03026639 , tol:  0.02741678
## epoch: 11
## epoch:  11 , error:  0.02963983 , tol:  0.0104591
## epoch: 12
## epoch:  12 , error:  0.02947078 , tol:  0.002859827
## epoch: 13
## epoch:  13 , error:  0.02915038 , tol:  0.005465555
## epoch: 14
## epoch:  14 , error:  0.02909747 , tol:  0.000908472
## epoch: 15
## epoch:  15 , error:  0.02895824 , tol:  0.00239811
## epoch: 16
## epoch:  16 , error:  0.02889019 , tol:  0.00117644
## epoch: 17
## epoch:  17 , error:  0.02869325 , tol:  0.003419965
## epoch: 18
## epoch:  18 , error:  0.02877655 , tol:  0.001449297
## epoch: 19
## epoch:  19 , error:  0.0288857 , tol:  0.001892921
## epoch: 20
## epoch:  20 , error:  0.02857388 , tol:  0.005426765
```

I have reconstructed the training_set to see how the auto encoder model performed. The model iterates over each column of the training_set and it encodes the input data by taking the element-wise maximum of the dot product between the training set column and the weights of the auto encoder. It then decodes the encoded representation by taking the element-wise maximum of the dot product between the encoded representation and the transposed auto encoder weights.

```r
# Reconstruct the test_set by looping through each column
reconstruct_train <- matrix(0, nrow(training_set), ncol(training_set))
colnames(reconstruct_train) <- colnames(training_set)
rownames(reconstruct_train) <- rownames(training_set)
for (i in 1:ncol(training_set)) {
  encoded <- pmax(training_set[, i] %*% autoencoder_model$w, 0)
  decoded <- pmax(encoded %*% t(autoencoder_model$w), 0)
  reconstruct_train[, i] <- decoded
}
```

I also calculated the Mean Absolute Error to see the mean absolute error between the training set and the reconstruct_train matrix. The MAE value showed 0.026 which I believed was good as lower the values indicating the error the better the reconstruction accuracy.

```r
# Calculating the mean absolute error
mae <- mean(abs(training_set - reconstruct_train))
mae
```

```
## [1] 0.02667287
```

Similarly, I also tried the autoencoder model in my test data as below. It performs the same function as above, howeve,r this time it is implemented in the test data to see how the model reconstructs the test dataset.

```r
library(ggplot2)

# create a dataframe with the original training set and the reconstructed training set
df <- data.frame(training_set = training_set[,1],
                 reconstruct_train = reconstruct_train[,1])

# create the scatterplot
ggplot(df, aes(x = training_set, y = reconstruct_train)) +
  geom_point() +
  ggtitle("Training Set vs Reconstructed Training Set") +
  xlab("Training Set") +
  ylab("Reconstructed Training Set")
```

```r
# Train the autoencoder model
reconstruct_test <- matrix(0, nrow(test_set), ncol(test_set))
colnames(reconstruct_test) <- colnames(test_set)
rownames(reconstruct_test) <- rownames(test_set)

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

  encoded <- pmax(test_set[, i] %*% autoencoder_model$w, 0)

  decoded <- pmax(encoded %*% t(autoencoder_model$w), 0)

  reconstruct_test[, i] <- decoded
}
```

I also calculated the MAE for the test dataset which showed the value of 0.0189 through which I came across that the model was performing well.

```r
# Calculating the mean absolute error
mae <- mean(abs(test_set - reconstruct_test))
mae
```

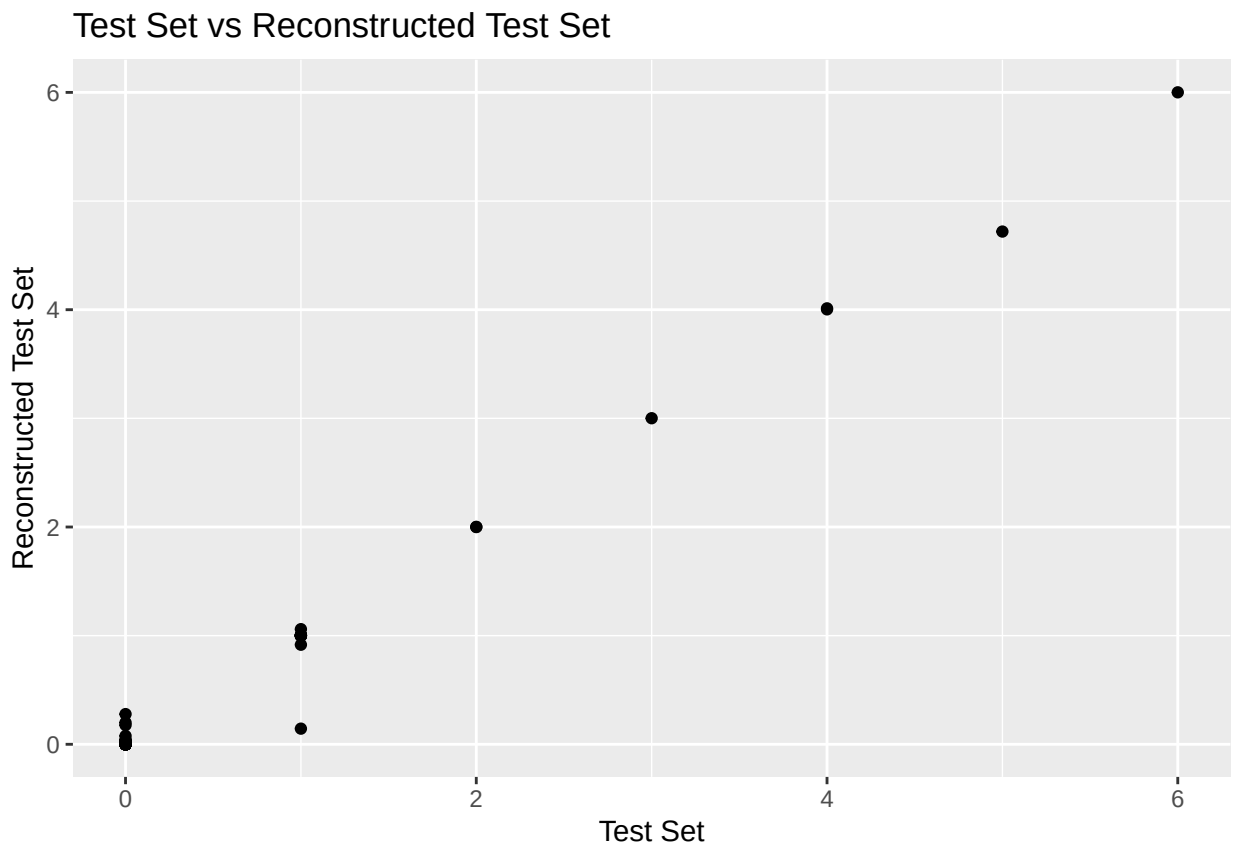
```
## [1] 0.01893003
```

Additionaly, I also tried to visualize it using the ggplot and through the plot we can see that data has been reconstructed in a linear relationship.

```r
library(ggplot2)

df <- data.frame(test_set = test_set[,1],
                 reconstruct_test = reconstruct_test[,1])

# creating the scatterplot
ggplot(df, aes(x = test_set, y = reconstruct_test)) +
  geom_point() +
  ggtitle("Test Set vs Reconstructed Test Set") +
  xlab("Test Set") +
  ylab("Reconstructed Test Set")
```



Normalization

Additionally, I also tried to normalize my data using min-max log normalization and z transform normalization and then again ran my autoencoder model, however, there wasn't much change after applying the normalization too. I also tried changing learning rate, epochs and the hidden layer, however, there wasn't much changes in the accuracy score.

```
# Function to calculate the min-max log normalization
normalize_data <- function(data) {
  return ((log(data + 1) - min(data)) / (max(data) - min(data)))
}

# Function to calculate the z transfrom normalization
normalize_z_transform <- function(data) {
  return ((log(data + 1) - mean(data)) / sd(data))
}

# Normalize the training data
training_set <- apply(training_set, 2, normalize_data)

# Normalize the test data
test_set <- apply(test_set, 2, normalize_data)
```

Here, the reconstruct_test is converted to a data.table object function. So, all the value that have non zero

values in the test_set object are set to 0 and it loops through each column of reconstruct data. I have used the order() function to determine the indices of the top 5 recommendations in descending order and the first top 5 elements are set to 1.

```
reconstruct_test[test_set != 0] <- 0


for (col in colnames(reconstruct_test)) {
  # Get top 5 recommendations for the column
  res <- rep(0, nrow(reconstruct_test))
  sorted_indices <- order(reconstruct_test[, col], decreasing = TRUE)

  if (length(sorted_indices) >= 5) {
    res[sorted_indices[1:5]] <- 1
  } else {
    res[sorted_indices] <- 1
  }

  reconstruct_test[, col] <- res
}
```

I then submitted the implemented auto encoder model to the Kaggle Challenge to see how the model performed and the error score I received was **0.05913** which was not an improvement from the previous score of ours which was at **0.04754**. I tried tweaking around with hyper parameter tuning, however, the best score I received with the auto encoder was at 0.05913.

```
table(colSums(reconstruct_test != 0))
```

```
##
##    5
## 6009
```

```
table(reconstruct_test)
```

```
## reconstruct_test
##      0      1
## 480720  30045
```

Submitting Recommendation

```
rownames(reconstruct_test) <- rownames(test_set)
submission <- reshape2::melt(reconstruct_test)
head(submission)
```

```
##                 Var1  Var2 value
## 1            Apapane test1     0
## 2     Hawaii_Elepaio test1     0
## 3     Kalij_Pheasant test1     0
## 4  Northern_Cardinal test1     0
## 5               Omao test1     0
## 6 Warbling_White_eye test1     0
```

Compare that to the submission format:

```
sample_submission <- read.csv("sample_submission.csv", row.names = 1)
submission_ids <- read.csv("submission_ids.csv", row.names = 1)
head(sample_submission)
```

```
##      Id Expected
## 1 row_1        0
## 2 row_2        1
## 3 row_3        0
## 4 row_4        0
## 5 row_5        0
## 6 row_6        0
```

```
head(submission_ids)
```

```
##      Id           row_name col_name
## 1 row_1            Apapane    test1
## 2 row_2     Hawaii_Elepaio    test1
## 3 row_3     Kalij_Pheasant    test1
## 4 row_4  Northern_Cardinal    test1
## 5 row_5               Omao    test1
## 6 row_6 Warbling_White_eye    test1
```

We can verify that our results line up with `submission_ids`:

```
all.equal(submission_ids$row_name, as.character(submission$Var1))
```

```
## [1] TRUE
```

```
all.equal(submission_ids$col_name, as.character(submission$Var2))
```

```
## [1] TRUE
```

This means that our `row_ Id` will map correctly. Now we just replace our recommendations with the sample submission:

```
sample_submission$Expected <- submission$value
```

Save this as a CSV:

```
write.csv(sample_submission, "submission.csv", row.names = FALSE)
```

Conclusion

The Kaggle Challenge 4 was interesting and challenging at the same time. I had expected that through using the autoencoder model the error score would improved than that of the last time, however, the output I received wasn't as per my expectation. The implementation of the autoencoder model was a great learning experience.

Reference

- Zachary DeBruine(2023, January). ML Challenge #1: Getting Started. Version: 6 from https://www.kaggle.com/code/zdebruine/getting-started