Math 208: Creating Neural Network Functions

Lily Samuel

2021-11-26

```
library(palmerpenguins)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(neuralnet)
```

```
##
## Attaching package: 'neuralnet'

## The following object is masked from 'package:dplyr':
##
##     compute
```

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v readr     2.1.4
## v ggplot2   3.4.2     v stringr   1.5.0
## v lubridate 1.9.2     v tibble    3.2.1
## v purrr     1.0.1     v tidyr     1.3.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x neuralnet::compute() masks dplyr::compute()
## x dplyr::filter()      masks stats::filter()
## x dplyr::lag()         masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
penguins_example <- penguins %>% drop_na %>% mutate(sex=ifelse(sex=="female",1,0))
```

Task 1:

write a function that requires three arguments: i. A data frame or tibble ii. A length 1 character vector indicating the name of the outcome column in the dataset. iii. A character vector of unspecified length containing the names of the input features to be selected and scaled. and returns a new data set which contains a tibble containing only the outcome vector which should be renamed outcome and the scaled feature vectors, each of which has been scaled using the scale function.

Solution:

```
outcome_func<-function(x_df, outcome, input_vec){
  outcome_1 <- x_df[,c(outcome, input_vec)] %>% mutate_at(~scale(.),.vars=vars(input_vec))
  names(outcome_1)[names(outcome_1)== outcome] <- "outcome"
  return(outcome_1)
  }

outcome_col<-c("sex")
z_vect_cols<-c("body_mass_g","bill_length_mm")
result_a<-outcome_func(penguins_example, outcome_col, z_vect_cols)
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(input_vec)
##
##   # Now:
##   data %>% select(all_of(input_vec))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
head(result_a)
```

```
## # A tibble: 6 x 3
##   outcome body_mass_g[,1] bill_length_mm[,1]
##     <dbl>           <dbl>              <dbl>
## 1       0          -0.568             -0.895
## 2       1          -0.506             -0.822
## 3       1          -1.19              -0.675
## 4       1          -0.940             -1.33
## 5       0          -0.692             -0.858
## 6       1          -0.723             -0.931
```

Task 2: Write a function to randomly split a data frame or tibble into Training and Test that requires two arguments: i. A data frame or tibble ii. The percentage of the total number of rows that should be from training and returns a list which has two elements, one that is the Training data and the other is the Test data.

Solution:

```r
split_data<-function(x_df, percentage){

split_labels <- sample(c("Training","Test"),
 prob=c(percentage,1-percentage), replace=T,
 size=nrow(x_df))

Training_sample <- x_df %>%
 filter(split_labels=="Training")

Test_sample <- x_df %>%
 filter(split_labels=="Test")
list(Training=Training_sample, Test=Test_sample)
}
result_b<-split_data(result_a,0.7)
glimpse(result_b)
```

```
## List of 2
##  $ Training: tibble [231 x 3] (S3: tbl_df/tbl/data.frame)
##   ..$ outcome      : num [1:231] 0 1 1 1 1 0 1 1 0 1 ...
##   ..$ body_mass_g  : num [1:231, 1] -0.568 -0.506 -1.189 -0.94 -1.251 ...
##   .. ..- attr(*, "scaled:center")= num 4207
##   .. ..- attr(*, "scaled:scale")= num 805
##   ..$ bill_length_mm: num [1:231, 1] -0.895 -0.822 -0.675 -1.334 -0.529 ...
##   .. ..- attr(*, "scaled:center")= num 44
##   .. ..- attr(*, "scaled:scale")= num 5.47
##  $ Test    : tibble [102 x 3] (S3: tbl_df/tbl/data.frame)
##   ..$ outcome      : num [1:102] 0 1 0 0 1 1 1 0 1 1 ...
##   ..$ body_mass_g  : num [1:102, 1] -0.692 -0.723 0.581 -0.506 -0.506 ...
##   .. ..- attr(*, "scaled:center")= num 4207
##   .. ..- attr(*, "scaled:scale")= num 805
##   ..$ bill_length_mm: num [1:102, 1] -0.858 -0.931 -0.876 -0.986 -1.59 ...
##   .. ..- attr(*, "scaled:center")= num 44
##   .. ..- attr(*, "scaled:scale")= num 5.47
```

```r
head(result_b)
```

```
## $Training
## # A tibble: 231 x 3
##    outcome body_mass_g[,1] bill_length_mm[,1]
##      <dbl>           <dbl>              <dbl>
##  1       0          -0.568             -0.895
##  2       1          -0.506             -0.822
##  3       1          -1.19              -0.675
##  4       1          -0.940             -1.33
##  5       1          -1.25              -0.529
##  6       0           0.240             -1.72
##  7       1          -0.630             -1.35
##  8       1          -0.940             -0.968
##  9       0           0.364             -0.273
## 10       1          -1.10              -1.75
## # i 221 more rows
##
```

```
## $Test
## # A tibble: 102 x 3
##    outcome body_mass_g[,1] bill_length_mm[,1]
##      <dbl>           <dbl>              <dbl>
##  1       0          -0.692             -0.858
##  2       1          -0.723             -0.931
##  3       0           0.581             -0.876
##  4       0          -0.506             -0.986
##  5       1          -0.506             -1.59
##  6       1          -1.25              -0.639
##  7       1          -1.13              -1.17
##  8       0           0.550             -0.767
##  9       1          -1.37              -1.46
## 10       1          -0.878             -0.803
## # i 92 more rows
```

Task 3:

Write a function that takes in the following arguments: i. A data frame or tibble with a column named outcome and other columns that are all scaled feature vectors ii. A vector of integers that can be used as the hidden argument to the neuralnet function, i.e. a list of numbers of nodes of the hidden layers of a neural network to return a neuralnet object that is the result of running the neuralnet function on the data frame/tibble with the hidden nodes specified from the second argument and the following other arguments: linear.output = FALSE, act.fct="logistic" and using the outcome variable as the outcome in the formula argument

```
fit_model<-function(x_df, hidden_nodes_vec){
 neural_net = neuralnet(outcome~.,linear.output = FALSE, act.fct="logistic",data=x_df, hidden=hidden_no
 return(neural_net)
}

outcome<-c("sex")
result_b[[1]]
```

```
## # A tibble: 231 x 3
##    outcome body_mass_g[,1] bill_length_mm[,1]
##      <dbl>           <dbl>              <dbl>
##  1       0          -0.568             -0.895
##  2       1          -0.506             -0.822
##  3       1          -1.19              -0.675
##  4       1          -0.940             -1.33
##  5       1          -1.25              -0.529
##  6       0           0.240             -1.72
##  7       1          -0.630             -1.35
##  8       1          -0.940             -0.968
##  9       0           0.364             -0.273
## 10       1          -1.10              -1.75
## # i 221 more rows
```

```
result_c<-fit_model(result_b[[1]], c(2,2))

plot(result_c)
```

Task 4:

Write a function that takes the following arguments: i. A neuralnet object ii. A data frame/tibble containing Training Data iii. A data frame/tibble containing Test Data and returns a vector containing the average training squared error and the average test squared error using the neuralnet object, where average squared error is as defined in the background section.

```
run_training_test<-function(model_obj, Training, Test){

train_predict <- predict(model_obj,newdata=Training)
Training_error<-Training %>% mutate(train_error_sq=(outcome-train_predict)^2) %>%
summarize(Avg_Error_train=mean(train_error_sq))

test_predict <- predict(model_obj,newdata=Test)
Test_error<-Test%>% mutate(test_error_sq=(outcome-test_predict)^2) %>%
summarize(Avg_Error_test=mean(test_error_sq))

c("Training_Error"=as.numeric(Training_error), "Test_Error"=as.numeric(Test_error))
}

run_training_test(result_c,result_b$Training,result_b$Test)
```

```
## Training_Error     Test_Error
##      0.1045217      0.1227034
```

Task 5:

Write a function that takes the following arguments: i. A data frame or tibble ii. A length 1 character vector indicating the name of the outcome column in the dataset. iii. A character vector of unspecified length containing the names of the input features to be selected and scaled. iv. The percentage of the total number of rows in the data/frame or tibble that should be used in the training data. and returns a tibble where each row contains the Average Training and Average Test squared error for fitting a two-layer neural network at all possible combinations of numbers of hidden nodes at each layer.

Solution:

```
final_func<-function(x_df, outcome, input_vec, percentage){
 Results<-as_tibble(expand.grid(`First layer`=c(1,2,3),`Second layer`=c(1,2,3), `Training error`=0, `Tes

 scaled_data<-outcome_func(x_df,outcome,input_vec)
 training_test_data<-split_data(scaled_data,percentage)

 for(i in 1:9){
 current_mod<-fit_model(training_test_data$Training,
 hidden_nodes_vec=as.numeric(Results[i,c("First layer","Second layer")])
)

 error<-run_training_test(current_mod,
 training_test_data$Training,
 training_test_data$Test)

 Results[i,c("Training error")]<-error[1]
 Results[i,c("Test error")]<-error[2]
 }
 Results
}
```

```r
set.seed(1)
my_results<-final_func(penguins_example,"sex", c("bill_length_mm","body_mass_g"), 0.7)
my_results
```

```
## # A tibble: 9 x 4
##   `First layer` `Second layer` `Training error` `Test error`
##           <dbl>          <dbl>            <dbl>        <dbl>
## 1             1              1           0.186         0.221
## 2             2              1           0.183         0.195
## 3             3              1           0.154         0.233
## 4             1              2           0.180         0.215
## 5             2              2           0.143         0.205
## 6             3              2           0.102         0.184
## 7             1              3           0.135         0.193
## 8             2              3           0.0955        0.177
## 9             3              3           0.0545        0.240
```