MATH 208 Assignment 3

The assignment contains one question **with 5 parts (a)-(e)**, each worth 10 points, for a total of 50 points. Your answers must be submitted in the form of a PDF and include both the answers to the question, along with your R code and output used to generate your answers.

The background

Neural networks have become a popular tool for analyzing datasets where the goal is to develop a complex prediction model which takes a set of input features and tries to predict the result of an outcome (or target) variable. These models work well in situations where the relationship between the input features and the outcome variable is highly nonlinear.

The basic structure of a neural network is as follows:

- We assume a set of K input features.
- We choose a number of hidden (unobserved) **layers**.
- For each layer, we choose a number of **nodes**.
- The inputs, layers, and outcome are connected by edges which have weights which need to be estimated.
- Every node also has its own bias node that is used to help adjust the linear combinations to improve prediction (similar to an intercept term in linear regression).
- Each node in each hidden layer contains a linear combination of values of previous nodes that then gets passed throughe the network to result in a final prediction.

Here is an example of how to fit such a network for the `palmerpenguins` data from the quizzes, where we try to predict the sex of the penguin from bill length and body mass.

```r
library(palmerpenguins) ## needs to be installed
library(neuralnet) ## needs to be installed
penguins_example <- penguins %>%
  drop_na %>%  #NA dropped
  # Make Females=1, Males=0
  mutate(sex=ifelse(sex=="female",1,0))
```

We will use this dataset for the rest of the assignment.

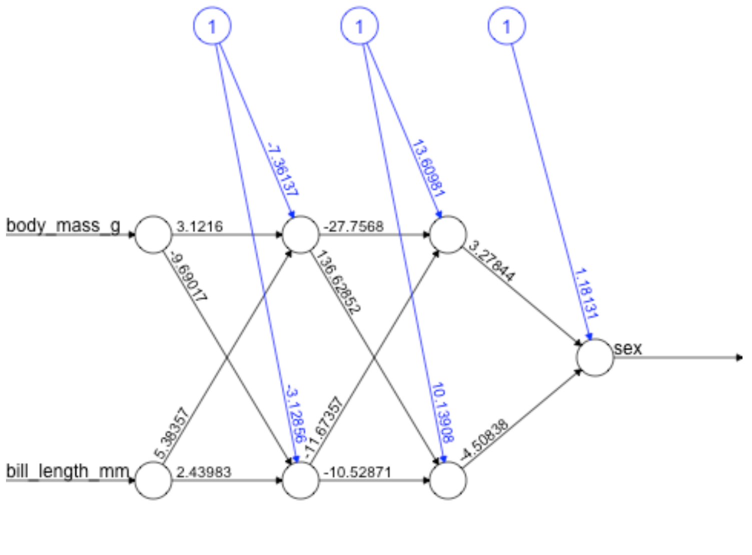
Additionally, the two input features need to be scaled to have average value 0 and standard deviation 1 in order to use the `neuralnet` function (from the `neuralnet` package) to fit the models without lots of extra work.

```r
penguins_with_bin_sex_no_na <-
  ## Select columns we need
  penguins_example[,c("sex",c("body_mass_g","bill_length_mm"))] %>%
  # columns scaled -- mutate at each in vars and save under the same name
  mutate_at(~scale(.),.vars=vars(c("body_mass_g","bill_length_mm")))
```

Now we can fit the neural network. Setting `linear.output=FALSE` and using the logistic function (argument of `act.funct`) converts the output of the neural network into a value between 0 and 1 which can be interpreted as a probability. The length of the vector of the `hidden` argument vector is the number of hidden layers. The value of each element of the `hidden` argument vector is the number of hidden nodes in that respective layer so `hidden=c(a,b)` means 2 hidden layers with `a` nodes in the first layer and `b` nodes in the second layer.

```
nn_penguins <- neuralnet(sex~body_mass_g+bill_length_mm,linear.output = FALSE,
                        act.fct="logistic",data=penguins_with_bin_sex_no_na, hidden=
c(2,2))
```

```
plot(nn_penguins)
```



Error: 26.803421 Steps: 1726

Above, we can see the structure of the network when there are two hidden layers and two hidden nodes per layer. The weights of the edges are in black, the blue edges and nodes are the bias terms. The weights and bias terms are chosen to minimize the sum of squared errors (shown above), i.e.

$$\sum_{i=1}^{\text{\# of units}} (y_i - \hat{y}_i)^2.$$

where y_i is the binary gender value for the penguin in row i of the data and $\hat{y}_i$ is the predicted probability from the neural network that this penguin in row i has $y_i = 1$. Other errors are possible in `neuralnet` , but the default is fine for this assignment. Note that a better measure is the average error per observation, which we will compute later.

In general, our neural network error tends to be under-estimated, because of the optimization of the weights. So we often split our data into a training sample and a test sample to evaluate the error independently from our estimated weights. I can do this randomly by doing:

```
# Note, I do this so you and I get the same results
# You do not have to set this for your assignment
set.seed(1101)

## The line above creates a vector with Training and test splits
## for each row of our data, approx. 80% train, 20% test
split_labels <- sample(c("Training","Test"),prob=c(0.8,0.2), replace=T,
                       size=nrow(penguins_with_bin_sex_no_na))
## How many of each kind?
table(split_labels)
```

```
split_labels
    Test Training
      62      271
```

```
# Create Training Data
Training_sample <- penguins_with_bin_sex_no_na %>%
  filter(split_labels=="Training")
nrow(Training_sample)
```

```
[1] 271
```

```
# Create Test Data
Test_sample <- penguins_with_bin_sex_no_na %>%
  filter(split_labels=="Test")
nrow(Test_sample)
```

```
[1] 62
```

Now I can fit the neural network to the training data and compute predictions and average squared error for the training data.

```
## Run neural network on training data
train_penguins <- neuralnet(sex~.,linear.output = FALSE,
                            act.fct="logistic",data=Training_sample, hidden=c(2,2))

## Compute predictions and error for training using predict function on neuralnet obj
ect

train_penguins_predict <- predict(train_penguins,newdata=Training_sample)
Training_sample %>% mutate(train_error_sq=(sex-train_penguins_predict)^2) %>%
  summarize(Avg_Error_train=mean(train_error_sq))
```

Avg_Error_train
<dbl>

0.171879

1 row

Now I compute the predictions and error for the test data.

```
## Compute predictions and error for test data using predict function on neuralnet ob
ject
test_penguins_predict <- predict(train_penguins,newdata=Test_sample)

## Compute test sums of squared error divided by number of test samples
Test_sample %>% mutate(test_error_sq=(sex-test_penguins_predict)^2) %>%
  summarize(Avg_error_test =mean(test_error_sq))
```

Avg_error_test
<dbl>
0.1853412

1 row

CONTINUED ON NEXT PAGE

The actual assignment

The goal of this assignment is to create a set of **functions** that will do ALL of the tasks in the code of the previous section for ANY dataset that contains a single outcome column and a set of possible input features. This is similar to how one might construct an R package to do all of the steps from before. **TAKE A DEEP BREATH**. I will break it all down for you.

a. First, 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. You CAN assume that the outcome column is already binary (contains 0 and 1). You should NOT assume that all of the columns in the original data frame or tibble will be used in the network, so you will need to choose the right ones using the appropriate argument. **Demonstrate that your function works by running it on the `penguins_example` tibble from the Background section for the two features used in the Background, `body_mass_g` and `bill_length_mm`.**

Solution:

```
wrangle_data_parta <- function(input_data,
                               outcome,
                               features){
 return_data <-
  ## Select columns we need
  input_data[,c(features)] %>%
   mutate(outcome=input_data %>% pull(outcome)) %>%
  # columns scaled -- mutate at each in vars and save under the same name
  mutate_at(~scale(.),.vars=vars(features))
return_data
}
result_a<-wrangle_data_parta(penguins_example,
                             "sex",
                             c("bill_length_mm","body_mass_g"))
head(result_a)
```

bill_length_mm	body_mass_g	outcome
<dbl>	<dbl>	<dbl>
-0.8946955	-0.5676206	0
-0.8215515	-0.5055254	1
-0.6752636	-1.1885721	1

-1.3335592	-0.9401915	1
-0.8581235	-0.6918109	0
-0.9312674	-0.7228585	1

6 rows

b. 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. **Demonstrate that your function works by running it on the tibble that you generated in part (a) with training fraction equal to 0.7**.

Solution::

```
split_data_partb<-function(input_data, percentage){

split_labels <- sample(c("Training","Test"),
                       prob=c(percentage,1-percentage), replace=T,
                       size=nrow(input_data))
# Create Training Data
Training_sample <- input_data %>%
  filter(split_labels=="Training")

# Create Test Data
Test_sample <- input_data %>%
  filter(split_labels=="Test")

list(Training=Training_sample, Test=Test_sample)
}

result_b<-split_data_partb(result_a,0.7)

glimpse(result_b)
```

```
List of 2
 $ Training: tbl_df [224 × 3] (S3: tbl_df/tbl/data.frame)
  ..$ bill_length_mm: num [1:224, 1] -0.822 -0.675 -0.858 -0.931 -0.876 ...
  .. ..- attr(*, "scaled:center")= num 44
  .. ..- attr(*, "scaled:scale")= num 5.47
  ..$ body_mass_g   : num [1:224, 1] -0.506 -1.189 -0.692 -0.723 0.581 ...
  .. ..- attr(*, "scaled:center")= num 4207
  .. ..- attr(*, "scaled:scale")= num 805
  ..$ outcome       : num [1:224] 1 1 0 1 0 1 0 0 1 0 ...
 $ Test    : tbl_df [109 × 3] (S3: tbl_df/tbl/data.frame)
  ..$ bill_length_mm: num [1:109, 1] -0.895 -1.334 -0.968 -0.273 -1.754 ...
  .. ..- attr(*, "scaled:center")= num 44
  .. ..- attr(*, "scaled:scale")= num 5.47
  ..$ body_mass_g   : num [1:109, 1] -0.568 -0.94 -0.94 0.364 -1.095 ...
  .. ..- attr(*, "scaled:center")= num 4207
  .. ..- attr(*, "scaled:scale")= num 805
  ..$ outcome       : num [1:109] 0 1 1 0 1 1 0 0 1 1 ...
```

c. 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. **Demonstrate that your function works by running it on the Training Data that you generated in part (b).**

Solution:

```
fit_model_partc<-function(input_data,hidden_vec){

  train_model <- neuralnet(outcome~.,linear.output = FALSE,
                           act.fct="logistic",data=input_data, hidden=hidden_vec)
  train_model

}
result_c<-fit_model_partc(result_b$Training, c(2,2))
plot(result_c)
```

d. 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. In other words, your function should compute both the average squared error for the Training data and the average squared error for the Test data, both using the `neuralnet` object to find the predictions. The vector returned by your function should be named with the first element named "Training_Error" and the second element named "Test_Error". **Demonstrate that your function works by running it on the Training and Test Data that you generated in part (b) and the `neuralnet` object from part (c).**

```
run_training_test_partd<-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_partd(result_c,result_b$Training,result_b$Test)
```

```
Training_Error     Test_Error
     0.1635680      0.1680435
```

e. 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 (1 through 3). Your returned tibble should look like this:

```
as_tibble(expand.grid(`First layer`=c(1,2,3),`Second layer`=c(1,2,3), `Training Error
`=NA, `Test error`=NA))
```

First layer	Second layer	Training Error	Test error
<dbl>	<dbl>	<lgl>	<lgl>

1	1	NA	NA
2	1	NA	NA
3	1	NA	NA
1	2	NA	NA
2	2	NA	NA
3	2	NA	NA
1	3	NA	NA
2	3	NA	NA
3	3	NA	NA

9 rows

where the NA's are replaced with the values for your runs. *Hint:* You can use the `expand.grid` function above to create a data.frame/tibble that you can iterate over the functions from parts (a) through (d) over the strategies that we have learned in class. Anything that works is acceptable (no need to optimize the speed). **Demonstrate that your function works by running it on the `penguins_example` tibble from the Background section for the two features used in the Background, `body_mass_g` and `bill_length_mm`.**

Solution:

```r
all_at_once_parte<-function(input_data,outcome,
                            features,percentage){

  Results<-as_tibble(expand.grid(`First layer`=c(1,2,3),`Second layer`=c(1,2,3), `Tra
ining error`=0, `Test error`=0))

  scaled_data<-wrangle_data_parta(input_data,outcome,features)
  training_test_data<-split_data_partb(scaled_data,percentage)

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

    error<-run_training_test_partd(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
}
set.seed(19010)
my_results<-all_at_once_parte(penguins_example,"sex",
                  c("bill_length_mm","body_mass_g"),
                  0.7)
my_results
```

First layer <dbl>	Second layer <dbl>	Training error <dbl>	Test error <dbl>
1	1	0.1978606	0.1824490
2	1	0.1977316	0.1832374
3	1	0.1188252	0.1282265
1	2	0.1754365	0.1628727
2	2	0.1830065	0.1944408
3	2	0.1506710	0.1913080
1	3	0.1977874	0.1822783
2	3	0.1342651	0.1523854
3	3	0.1178462	0.1285008

9 rows