Lab 19 MATH 4322

Neural Network

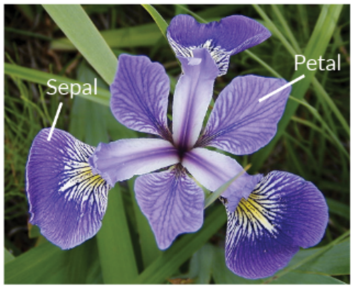
Fall 2022

Instructions

- You can print this out and write on this or write/type on a separate sheet. I also provide a Rmarkdown version of this if you desire.
- Upload your answers in BlackBoard as you do with your homework.
- The questions are in red.
- This lab we will work on the `iris` data.
- We will use the `Keras` library to help build a neural network. Installation of the package is here https://hastie.su.domains/ISLR2/keras-instructions.html.
- You will also need to install the miniconda installation of python. This is where I went to install miniconda: https://docs.conda.io/en/latest/miniconda.html.
- Also you will need to install the package `corrplot` if you have not done so.

Exploratory Data Analysis

There are three species of iris flowers: versiolor, setosa and virginica.



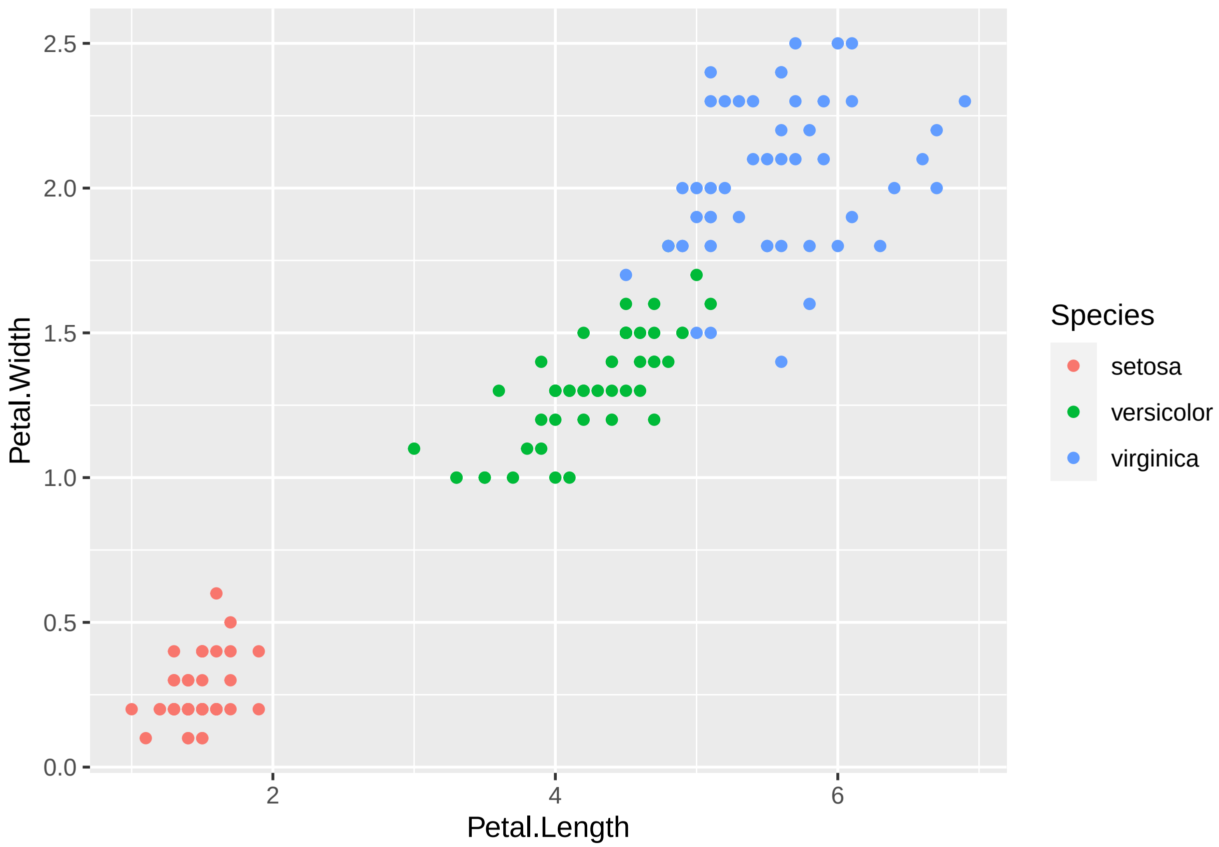
Iris Versicolor

Iris Setosa

Iris Virginica

For a comparison of the three iris species with respect to `Petal.Length` and `Petal.Width`, we can perform a scatterplot.

```
library(ggplot2)
ggplot(iris,aes(x = Petal.Length, y = Petal.Width, color = Species)) +
  geom_point()
```

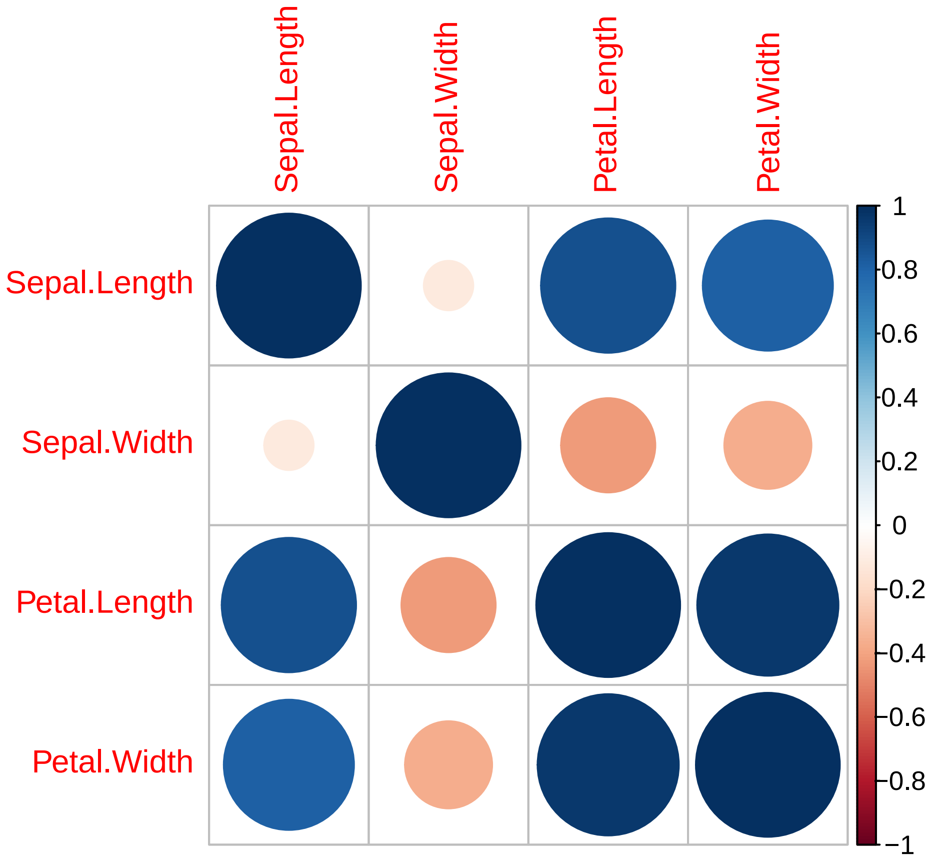


Question 1: Does there seem to be a difference in the measurements for each of the species of iris?

Yes

Next we can look at the correlations of the variables with a correlation plot.

```
M = cor(iris[,1:4]) #Store the overall correlation matrix in M.
library(corrplot)
corrplot(M, method = "circle")
```



Question 2: Which variables are highly correlated?

Petal Length wit Petal Width

Separate into Traning and Test subsets

Randomly divide the data into training and testing subset.

```
n <- nrow(iris)
p <- ncol(iris)-1

set.seed(1)
train <- sample(n, 0.8*n)

train_data <- iris[train, 1:p]
train_labels <- iris[train, p+1]

test_data <- iris[-train, 1:p]
test_labels <- iris[-train, p+1]
```

Random Forest

We will look at the random forest to predict the species. Then compare how well this works to the neural network.

```
library(randomForest)
set.seed(10)
rf.iris = randomForest(Species ~., data = iris,
                         subset = train,
                         mtry = sqrt(p),
                         importance = TRUE)
yhat.rf = predict(rf.iris,newdata = iris[-train,],type = "response")
table(yhat.rf,iris[-train,]$Species)
```

Question 3: What is the test error rate?

6.67%

Normalizing the data

For fit the neural network, it is best to scale the data so that each column has mean zero and variance one. This is called **Normalizing** the data.

```
train_data <- scale(train_data)
# Use means and standard deviations from training set to normalize test set
col_means_train <- attr(train_data, "scaled:center")
col_stddevs_train <- attr(train_data, "scaled:scale")

test_data <- scale(test_data,
                   center = col_means_train,
                   scale = col_stddevs_train)
```

One-Hot Encoding for the Categorical Reponse

When you want to model multi-class classification problems with the help of ANN, it is necessary to convert the response factor variable into numeric representation by one-hot encoding (OHI):.

- our response `Species` can take on values "setosa", "versicolor", "virginica"
- We represent it via a **numerical** vector $\mathbf{x} = (x_1, x_2, x_3)$, such that

Species	x_1	x_2	x_3
"setosa"	1	0	0
"versicolor"	0	1	0
"virginica"	0	0	1

To do this, *keras* suggests function *to_categorical*():

```
library(keras)
train_labels <- to_categorical(as.numeric(train_labels)-1)
head(train_labels)
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    0    0    1
## [3,]    1    0    0
## [4,]    1    0    0
## [5,]    0    1    0
## [6,]    0    1    0
```

```
test_labels <- to_categorical(as.numeric(test_labels)-1)
```

Next we want to set the seed for reproducibility.

```
tensorflow::tf$random$set_seed(1)
```

Building the Model with *Keras*

To start constructing a model, you should first initialize a sequential model with the help of the *keras_model_sequential()* function. We plan on constructing a network with

- a single hidden layer, which is fully-connected (layer_dense) and has 4 *ReLU* nodes (*units* = 4, *activation* = "relu"); the # of nodes for hidden layer is calculated via the rule of thumb

$$\#\text{ nodes in hidden layer} = \frac{\#\text{ nodes in input layer} + \#\text{ nodes in output layer}}{2} = \frac{4+3}{2} \approx 4$$

- a 3-node fully-connected output layer, which applies a softmax function.

```
library(embed) #https://github.com/tidymodels/embed
model <- keras_model_sequential(layers = list(
  layer_dense(units = 4, activation = 'relu', input_shape = dim(train_data)[2]),
  layer_dense(units = 3, activation = 'softmax')
))

model
```

(4+1)*4 (1 for bias)

```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## dense (Dense)                       (None, 4)                       20
## ________________________________________________________________________________
## dense_1 (Dense)                     (None, 3)                       15
## ================================================================================
## Total params: 35
## Trainable params: 35
## Non-trainable params: 0
## ________________________________________________________________________________
```

Compile and Train the Model

We need to compile the model by specifying:

- loss function, *loss* - what criteria does our ANN optimize?
- optimizing algorithm, *optimizer* - how do we optimize (details left out),
- model performance metrics, *metrics*.

```
compile(model,
  loss = 'categorical_crossentropy',
  optimizer = 'adam',
  metrics = 'accuracy'
)
```

Since we have a classification problem, here the metric is the overall prediction accuracy which is:

$$accuracy = \frac{\#\text{ of correct predictions}}{\#\text{ of all predictions}}$$

Next, we proceed to train, or fit, the model while using our training subset. We will train it

- for 200 epochs,

- feeding $batch\_size = 32$ input samples at a time,

- holding out 20% of the training data to be used for out-of-sample validation of the training process. This validation set will provide us with ability to track model's accuracy performance on data that wasn't used for training.

```
# Store the fitting history in `history`
history <- fit(model,
               train_data, train_labels,
               epochs = 200,
               batch_size = 32,
               validation_split = 0.2)
```

When $verbose = 0$ option is not specified, the progress printouts will be showing up after each epoch. Those will include:

- the loss and accuracy values on 96 training samples (out of 120 total training samples),
- the loss and accuracy values on 24 validation samples (out of 120 training samples).

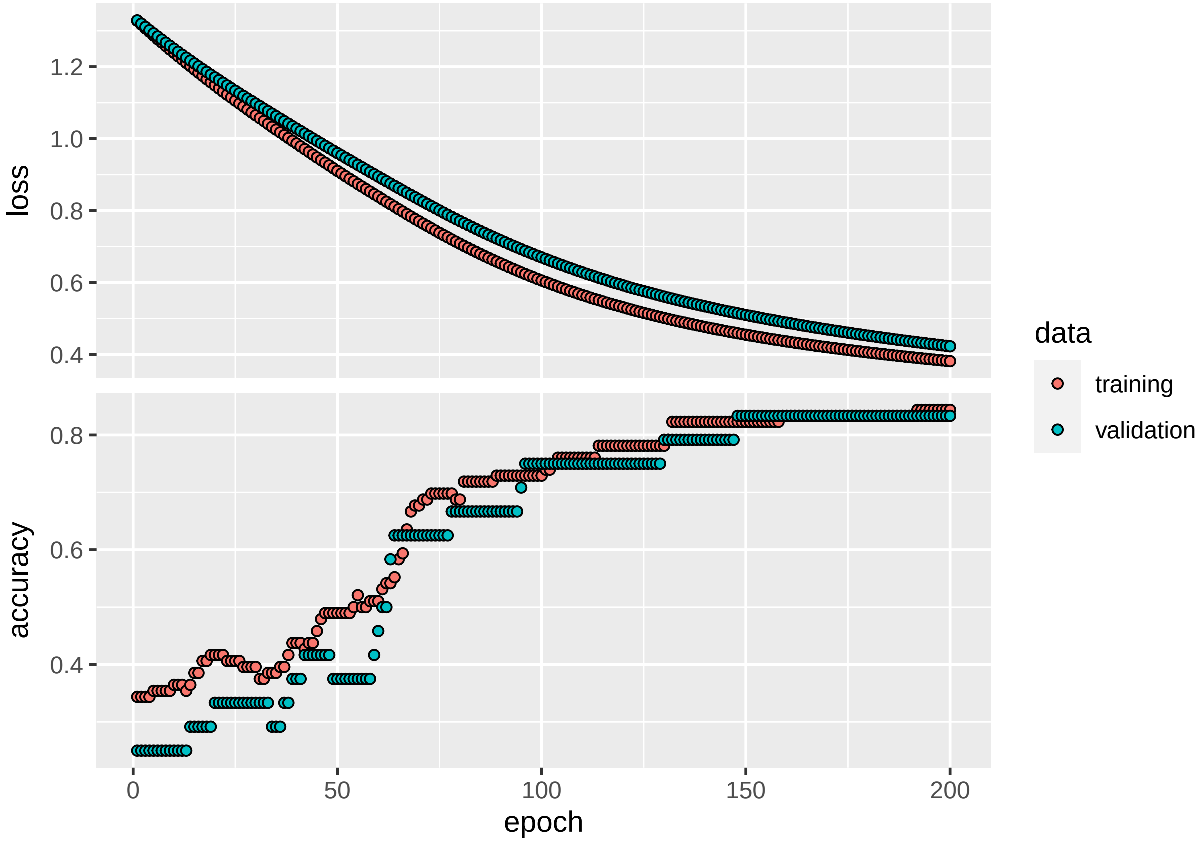
```
history
```

Question 4: What is the accuracy for the validation data?

83.33%

We can also plot this:

```
plot(history, smooth=F)
```

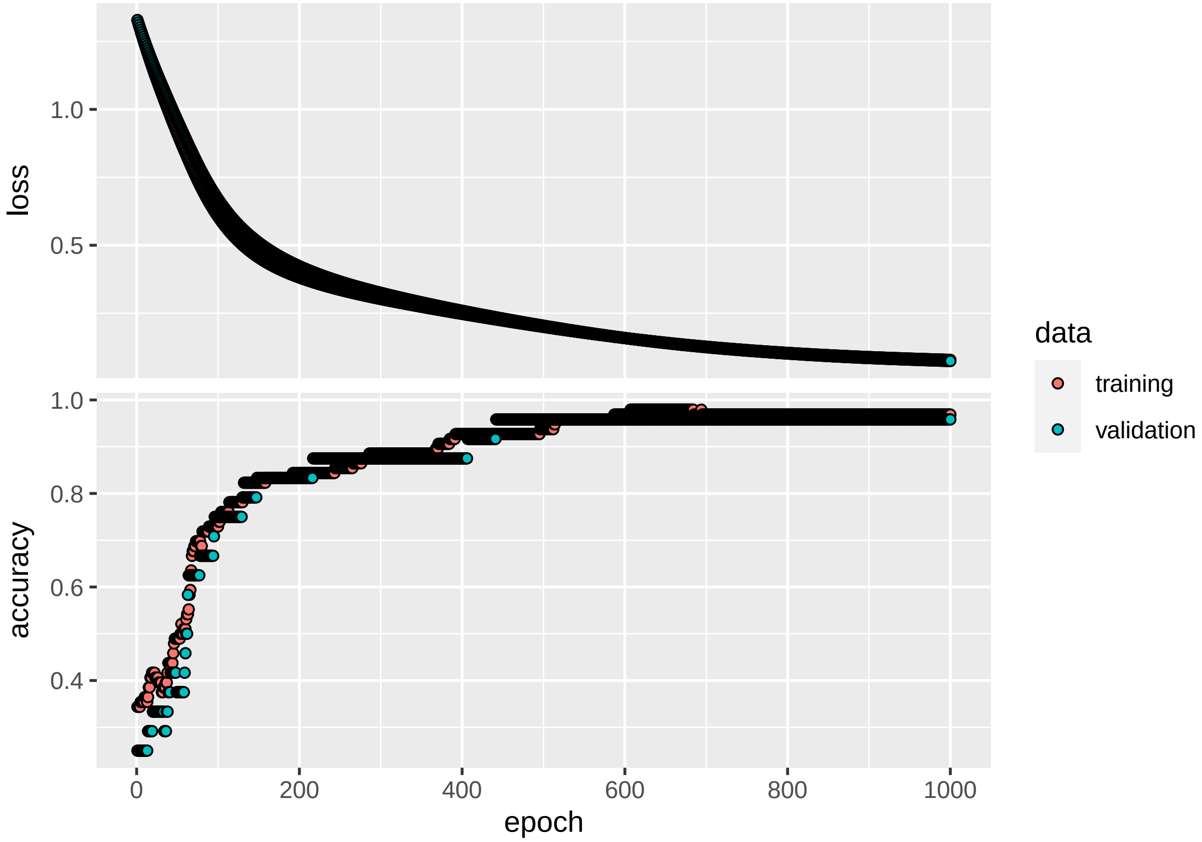


UPDATE: MORE EPOCHS and EARLY STOPPING

To get better accuracy we could use more epochs. Here we will use 1000 epochs and prevent overfiiting by early stopping after 20 epochs of no improvement to the training cross-entropy loss.

```
#use_session_with_seed(1)
tensorflow::tf$random$set_seed(1)

## Early stopping callback function
early_stop <- callback_early_stopping(monitor = "val_loss",
                                      patience = 20)

#### Initializing & Building the  model with KERAS

# Initialize a sequential model
model <- keras_model_sequential(layers = list(
  layer_dense(units = 4, activation = 'relu', input_shape = dim(train_data)[2]),
  layer_dense(units = 3, activation = 'softmax')
))

model



### COMPILE the model.
```

```
# Compiling the model

compile(model,
        loss = 'categorical_crossentropy',
        optimizer = 'adam',
        metrics = 'accuracy'
)



### FIT THE MODEL

# Store the fitting history in `history`
history <- fit(model,
               train_data, train_labels,
               epochs = 1000,
               batch_size = 32,
               validation_split = 0.2,
               callbacks=list(early_stop))


# Plot the history
plot(history, smooth=F)
```



```
# Print the training loss and metrics
history
```

Question 5: What is the validation accuracy after 1000 epochs?

98.83%

Test Data Predictions

We can use the model to predict the species for the test subset.

```
test.pred = predict(model,test_data)
head(test.pred,2)
tail(test.pred,2)
```

This provides the calculated class probabilities for each species.

We can put these predictions into a confusion matrix.

```
test_labels_num <- as.numeric(iris[-train, p+1] )-1
model %>% predict(test_data) %>% k_argmax()

# Evaluate on test data and labels
evaluate(model,
         test_data, test_labels)
```

Question 6: What is the accuracy rate of the test data with this model?

100%