## Developing a Deep Learning Model

Student Name:

Student ID:

**Ways for importing necessary libraries in deep learning:**

* For analysis and data manipulation

import numpy as np

import pandas as pd

* For deep learning frameworks

import tensorflow as tf

import torch

* For machine learning utilities

from sklearn.model\_selection import test\_train\_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

* For data visualization

import seaborn as sns

import matplotlib as plt

* For image processing

import cv2

* For natural language processing

import spacy

import nltk

Code for deriving the summary of species

Code for finding mean and standard deviation:

**For mean:**

*Load the necessary library*

library(readr)

*Loading the dataset*

iris\_data <- read\_csv("path/to/your/iris\_exams.csv")

*For calculating the mean of Sepal.Length*

mean\_sepal\_length <- mean(iris\_data\_Sepal.Length,

na.rm = TRUE)

*Results*

print(mean of sepal.length\_sepal\_length)

**For standard deviation:**

*Loading necessary library*

library(readr)

*Loading dataset*

iris\_data <- read\_csv("path/to/your/iris\_exams.csv")

*Calculating the standard deviation of Sepal.Length*

sepal\_length\_std\_dev <- sd(iris\_data\_Sepal.Length,

na.rm = TRUE)

### Checking the assumptions for additivity, linearity, normality, homogeneity and Homoscedasticity

**For Additivity**

Below is the R code for finding the additivity in the dataset

*Loading necessary libraries*

library(species)

*Loading dataset of species*

data(species)

Fitting linear model without interaction terms

model\_no\_interaction <- lm(sepal.length + sepal width + petal width + petal length, data = species)

Fitting linear model with interaction terms

model\_with\_interaction <- lm(sepal.length + sepal width + petal width + petal length, data = species)

Comparing models using ANOVA

anova\_results <- anova(model\_no\_interaction, model\_with\_interaction)

Display ANOVA results

print(anova\_results)

**Code for linearity:**

*Load necessary libraries*

library(species)

*Load species dataset*

data(species)

*Fit linear model*

model <- lm(sepal.length + sepal width + petal width + petal length, data = species)

plot(model)

**Code for** **Normality:**

*Loading necessary libraries*

library(species)

*Loading species dataset*

data(species)

*Fitting a linear model*

model <- lm(sepal.length + sepal width + petal width + petal length, data = species)

*Extracting residuals*

residuals <- model$residuals

*Performing* normality test using *Shapiro-Wilk*

shapiro\_test <- shapiro.test(residuals)

**Code for Homogeneity:**

*Load necessary libraries*

library(species)

library(MASS)

*Load speciesdataset*

data(species)

*Fitting the data into a simple linear model*

model <- lm (sepal length + sepal width + petal length + petal width, data = species)

*Extract the residuals*

residuals <- model$residuals

*Perform the Levene's test*

levene\_test <- leveneTest(residuals ~ factor(mtcars$cyl), data = species)

*Perform the Bartlett's*

bartlett\_test <- bartlett.test(residuals ~ factor(mtcars$cyl), data = species)

*Display the results*

print(levene\_test)

print(bartlett\_test)

In the output we get:

Bartlett's: p-value = 0.6112

**Code for Homoscedasticity**

*Load the necessary libraries*

library(species)

library(lmtest)

*Load the species dataset*

data(species)

*For performing the Breusch pagan test*

bp\_test <- bp\_test(model)

*Printing the results of Breusch pagan test*

print(bp\_test)

### Accuracy

**Code for Knowing the accuracy of model:**

import tensorflow as tf

from tensorflow.keras.models import sequential

from tensorflow.keras.layers import dense

For example in a dataset

For training

x\_train,

x\_train,

For testing

x\_test,

y\_test,

For building a simple model

model = Sequential([

dense(60,

activation='relu',

input\_shape=(inputshape,)),

dense(60,

activation='relu'),

dense(num\_classes,

activation='softmaximum')

])

Compiling the deep learning model

model.compiler(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy of model'])

Training the model

model.fit(x\_trained,

y\_trained,

epochs of model = 10,

splitting\_validation = 0.3)

Evaluating the model

loss,

accuracy of model = model.evaluation(X\_testing,

y\_testing)

print(f'Accuracy: {accuracy of model \* 100:.2f}%')

### Precision

**Code for Precision of a Model**

Here is the code written for building a simple model

import numpy as num

import tensorflow as tenfl

from tensorflow.keras.model import sequential

from tensorflow.keras.layer import dense

from sklearn.metrics import precision\_scores

For building a simple model

simple\_model = Sequential([

dense(60, activation='rel',

input\_shape=(input\_shape,)),

dense(60,

activation='rel'),

dense(num.classes,

activation='soft\_max')

])

For compiling model

model.compile(optimizer of model = 'adam',

loss during compiling = 'categorical\_crossentropy',

metrics of model = ['accuracy'])

For training model

model.fitting(x\_training,

y\_training,

Epochs during training = 10,

validation\_splitting = 0.2)

For making predictions

y\_pred\_prob = model.prediction (x\_test)

y\_pred = np.argmax(y\_prediction\_prob,

axis = 1)

y\_true = np.argmax(y\_test,

axis=1)

For calculating the precision

precision = precision\_score(y\_true, y\_prediction,

average = 'weighted')

print(f'Precision gained: {precision gained\* 100:.2f}%')

### F1 Score

**Code for knowing the F1 score**

Below mentioned is the code for F1 Score

import numpy as num

import tensorflow as tensfl

from tensorflow\_keras\_models importing sequential

from sklearn\_metrics\_import f1\_score

from tensorflow\_keras\_layers\_import dense

For building a simple deep model

model = Sequential([

dense(60, activation = 'rel',

input\_shape=(input.shape,)),

dense(60,

activation='rel'),

Dense(num.classes, activation= 'softmax')

])

For compiling the model

model.compiler(optimizer = 'adam',

loss at compiling = 'categorical.crossentropy',

metrics= ['accuracy of model'])

For training the model

model.fit(X data\_training,

Y data\_training,

epochs while training = 10,

validation.splitting = 0.2)

For making predictions

y\_pred\_prob = model.prediction(X\_test)

y\_pred = np.argmax(y\_prediction\_prob,

axis=1)

y\_true = np.argmax(y\_testing,

axis=1)

For calculating F1 score

f1 = f1\_score(y\_true,

y\_pred,

average value of F1 score = 'weighted')

print(f'F1 Score of data: {f1 \* 100:.2f}%')

### For handling missing values

Code for handling missing values

data = lm (rchisq (nrow (data.no\_missing),6),

data\_nomissing

Summary of data

Call:

lm(formula = rchisq (nrow (data.no\_missing),6),

data = data.no\_missing)