**Experiment 1**

**Write a program Print Dimensions of dataset**

*Code:*

import pandas as pd

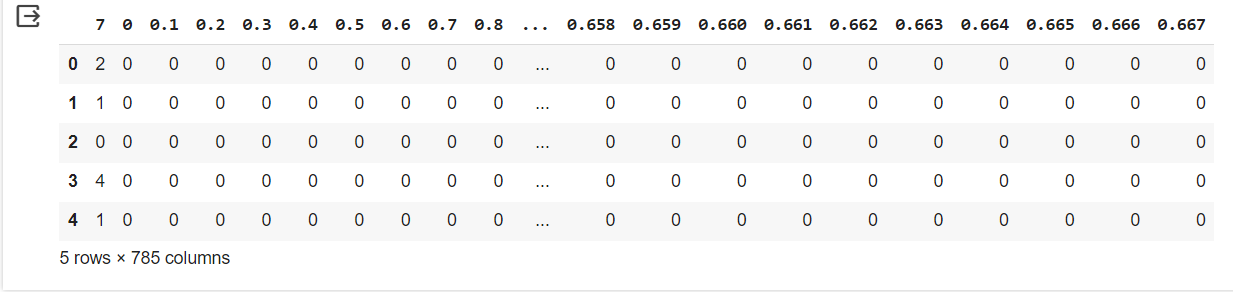
df = pd.read\_csv("/content/sample\_data/mnist\_test.csv")

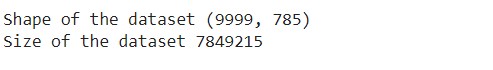
print(df.head()) *#returns top 5 rows/tuples*

print("Shape of the dataset",df.shape) *#returns the shape (dimensions) of the datase*t

print("Size of the dataset",df.size) *#returns the total number of cells*

*Output:*





**Experiment 2**

**Write a program to Calculate of Accuracy Values.**

*Code:*

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.datasets import load\_iris

from sklearn.metrics import accuracy\_score

# Loading the dataset

X, Y = load\_iris(return\_X\_y = True)

# Splitting the dataset in training and test data

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.3, random\_state = 0)

# Training the model using the Support Vector Classification class of sklearn

svc = SVC()

svc.fit(X\_train, Y\_train)

# Computing the accuracy score of the model

Y\_pred = svc.predict(X\_test)

score = accuracy\_score(Y\_test, Y\_pred)

print("Accuracy Score :",score)

*Output:*



**Experiment 3**

**Write a program to Build an Artificial Neural Network Classifier**

*Code:*

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.datasets import load\_iris

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

#preprocess the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

#split the data into training and testing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state = 42)

# Build the neural network model

model = tf.keras.Sequential([

tf.keras.layers.Dense(10,activation ='relu', input\_shape=(X\_train.shape[1],)),

tf.keras.layers.Dense(1,activation='sigmoid')

])

model.compile(optimizer ='adam',loss='binary\_crossentropy',metrics=['accuracy'])

model.fit(X\_train,y\_train,epochs=50,batch\_size=32,validation\_split=0.2)

y\_pred = model.predict(X\_test)

y\_pred\_binary = (y\_pred > 0.5).astype(int)

#Calculate accuracy

acc = accuracy\_score(y\_test, y\_pred\_binary)

print(acc)

*Output:*



**Experiment 4**

**Write a program to Compose Matrix Shape and Tensor Shape**

*Code:*

import tensorflow as tf

def compose\_matrix\_shape(matrix\_shape):

    return tuple(matrix\_shape)

def compose\_tensor\_shape(tensor\_shape):

    return tf.TensorShape(tensor\_shape)

# Compose matrix shape

matrix\_shape = [3, 4]

composed\_matrix\_shape = compose\_matrix\_shape(matrix\_shape)

print("Composed Matrix Shape:", composed\_matrix\_shape)

# Compose tensor shape

tensor\_shape = [None, 5, 5]

composed\_tensor\_shape = compose\_tensor\_shape(tensor\_shape)

print("Composed Tensor Shape:", composed\_tensor\_shape)

*Output:*

****

**Experiment 5**

**Write a program to showing accessing and manipulation of tensors.**

*Code:*

import tensorflow as tf

# Create a tensor

tensor = tf.constant([[1, 2, 3], [4, 5, 6]])

# Accessing elements of the tensor

print("Tensor:")

print(tensor)

print("Shape:", tensor.shape)

print("Data type:", tensor.dtype)

print("Number of dimensions:", tensor.ndim)

# Accessing specific elements

print("Accessing specific elements:")

print("Element at (0, 0):", tensor[0, 0].numpy())

print("Element at (1, 2):", tensor[1, 2].numpy())

# Manipulating tensors

# Addition

tensor\_add = tensor + 10

print("Tensor + 10:")

print(tensor\_add)

# Multiplication

tensor\_mul = tensor \* 2

print("Tensor \* 2:")

print(tensor\_mul)

# Reshaping tensor

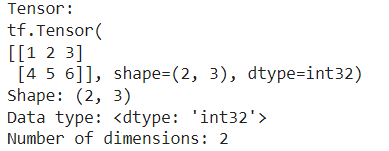
tensor\_reshaped = tf.reshape(tensor, (3, 2))

print("Reshaped tensor (3x2):")

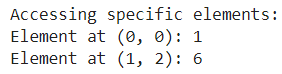
print(tensor\_reshaped)

*Output:*

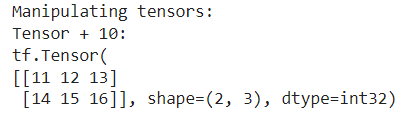
**# Accessing elements of the tensor**

****

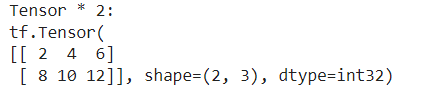
**# Accessing specific elements**

****

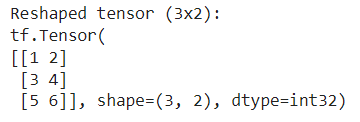
**# Manipulating tensors – Addition**

****

**# Multiplication**

****

**# Reshaping tensor**

****

**Experiment 6**

**Write a program to understand the mechanism of practically training a binary classifier**

*Code:*

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import make\_classification

# Generate synthetic data

X, y = make\_classification(n\_samples=1000, n\_features=20, n\_classes=2, random\_state=42)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define the model

model = tf.keras.Sequential([

tf.keras.layers.Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)),

tf.keras.layers.Dense(32, activation='relu'),

tf.keras.layers.Dense(1, activation='sigmoid')

])

# Compile the model

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

# Train the model

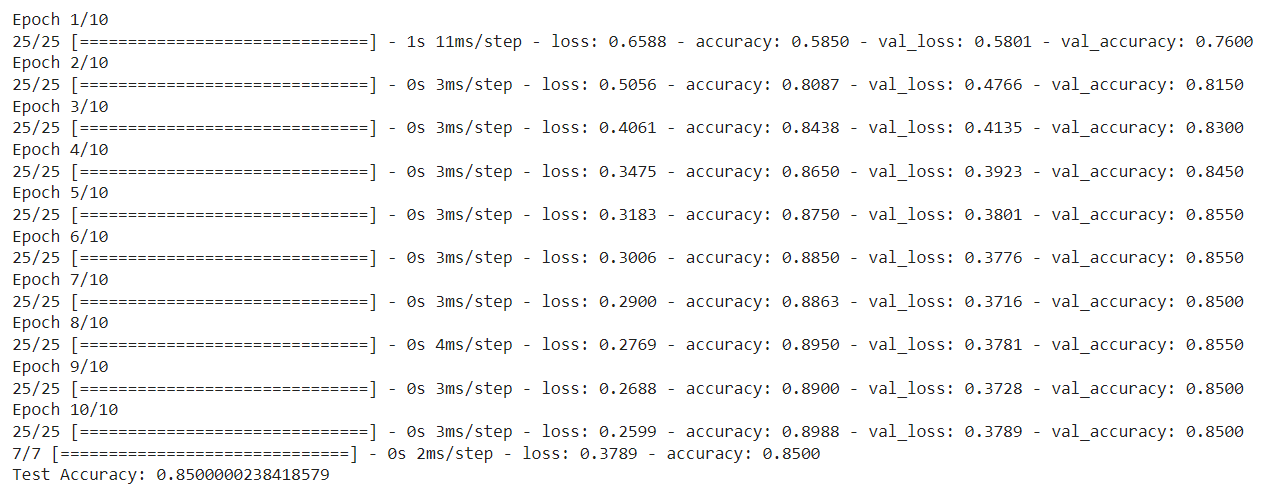
history = model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_test, y\_test))

# Evaluate the model

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)

print("Test Accuracy:", test\_accuracy)

*Output:*

****

**Experiment 7**

**Write a program to show regression Data sampling.**

*Code:*

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_regression

# Generate synthetic regression data

X, y = make\_regression(n\_samples=100, n\_features=1, noise=5, random\_state=42)

# Plot the data

plt.scatter(X, y, color='blue')

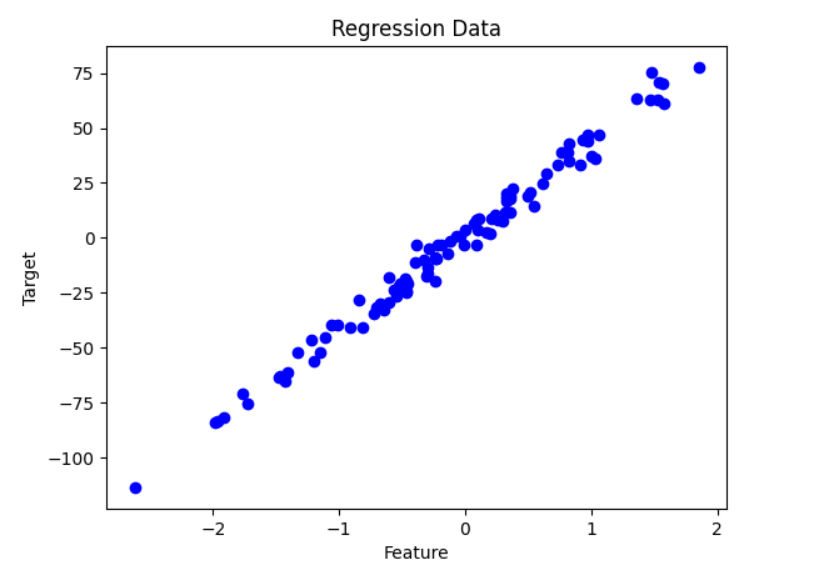
plt.title('Regression Data')

plt.xlabel('Feature')

plt.ylabel('Target')

plt.show()

*Output:*

****

**Experiment 8**

**Write a program to Combat Overfitting**

*Code:*

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.regularizers import l2

from sklearn.datasets import make\_classification

from sklearn.model\_selection import train\_test\_split

# Generate synthetic data

X, y = make\_classification(n\_samples=1000, n\_features=20, n\_classes=2, random\_state=42)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define a function to create a model

def create\_model(input\_dim, regularizer=None, dropout\_rate=None):

model = Sequential()

model.add(Dense(64, activation='relu', input\_dim=input\_dim, kernel\_regularizer=regularizer))

if dropout\_rate:

model.add(Dropout(dropout\_rate))

model.add(Dense(32, activation='relu', kernel\_regularizer=regularizer))

if dropout\_rate:

model.add(Dropout(dropout\_rate))

model.add(Dense(1, activation='sigmoid'))

return model

# Baseline model

print("Baseline Model:")

baseline\_model = create\_model(X\_train.shape[1])

baseline\_model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

baseline\_history = baseline\_model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test))

# Regularization - L2

print("\nModel with L2 Regularization:")

l2\_model = create\_model(X\_train.shape[1], regularizer=l2(0.001))

l2\_model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

l2\_history = l2\_model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test))

# Dropout

print("\nModel with Dropout:")

dropout\_model = create\_model(X\_train.shape[1], dropout\_rate=0.2)

dropout\_model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

dropout\_history = dropout\_model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test))

*Output:*

# Baseline model

**

# Regularization - L2

**

# Dropout

**

**Experiment 9**

**Write a program to build a simple autoencoder based on a fully connected layer in keras.**

*Code:*

import numpy as np

import tensorflow as tf

from tensorflow.keras.layers import Input, Dense

from tensorflow.keras.models import Model

from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt

# Load and preprocess the dataset

(x\_train, \_), (x\_test, \_) = mnist.load\_data()

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

x\_train = np.reshape(x\_train, (len(x\_train), np.prod(x\_train.shape[1:])))

x\_test = np.reshape(x\_test, (len(x\_test), np.prod(x\_test.shape[1:])))

# Define the autoencoder architecture

input\_dim = x\_train.shape[1]

encoding\_dim = 32

input\_img = Input(shape=(input\_dim,))

encoded = Dense(encoding\_dim, activation='relu')(input\_img)

decoded = Dense(input\_dim, activation='sigmoid')(encoded)

autoencoder = Model(input\_img, decoded)

# Compile the autoencoder

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

# Train the autoencoder

autoencoder.fit(x\_train, x\_train,

epochs=50,

batch\_size=256,

shuffle=True,

validation\_data=(x\_test, x\_test))

# Encode and decode some digits

encoded\_imgs = autoencoder.predict(x\_test)

decoded\_imgs = autoencoder.predict(x\_test)

# Plot original and reconstructed images

n = 10

plt.figure(figsize=(20, 4))

for i in range(n):

# Original images

ax = plt.subplot(2, n, i + 1)

plt.imshow(x\_test[i].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

# Reconstructed images

ax = plt.subplot(2, n, i + 1 + n)

plt.imshow(decoded\_imgs[i].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

plt.show()

*Output:*

****

**Experiment 10**

**Write a program to Build an Convolutional Neural Network**

*Code:*

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import models, layers, optimizers, losses, metrics

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation = 'relu', input\_shape=(28, 28, 1)))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.summary()

model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(10, activation='softmax'))

model.summary()

#Load dataset and preprocess data

(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()

train\_images = train\_images.reshape((60000, 28, 28, 1))

train\_images = train\_images.astype('float32') / 255

test\_images = test\_images.reshape((10000, 28, 28, 1))

test\_images = test\_images.astype('float32') / 255

plt.imshow(train\_images[59999])

train\_labels = to\_categorical(train\_labels)

test\_labels = to\_categorical(test\_labels)

#compile and train model

model.compile(

optimizer='rmsprop',

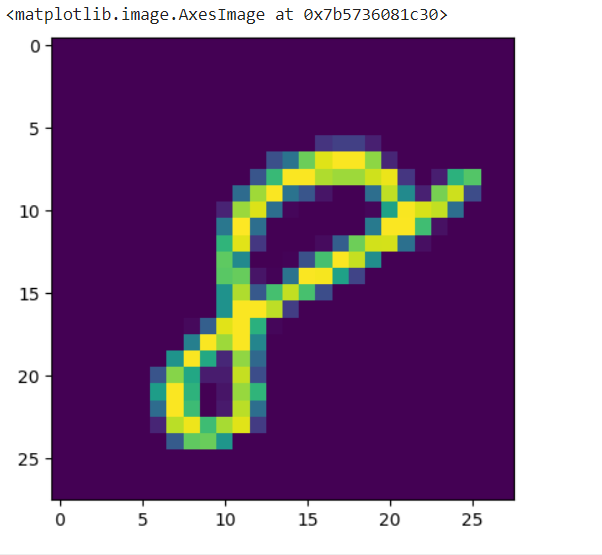
loss='categorical\_crossentropy',

metrics = ['accuracy']

)

model.fit(train\_images, train\_labels, epochs=5, batch\_size=64)

model.evaluate(test\_images, test\_labels)



**Experiment 11**

**Program for Multi-Classification using MNIST Dataset**

*Code:*

import tensorflow as tf

import keras

from keras import Sequential

from keras.layers import Dense, Flatten

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.mnist.load\_data()

print(x\_train.shape)

print(y\_train.shape)

print(x\_test.shape)

print(y\_test.shape)

import matplotlib.pyplot as plt

plt.imshow(x\_train[4])

x\_train = x\_train/255

x\_test = x\_test/255

model=Sequential()

model.add(Flatten(input\_shape = (28,28)))

model.add(Dense(128, activation='relu'))

model.add(Dense(32,activation='relu'))

model.add(Dense(10,activation='softmax'))

model.summary()

model.compile(loss='sparse\_categorical\_crossentropy', optimizer='Adam', metrics = ['accuracy'])

history = model.fit(x\_train, y\_train, epochs=25, validation\_split = 0.2)

y\_prob = model.predict(x\_test)

y\_pred = y\_prob.argmax(axis = 1)

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test, y\_pred)

*Output:*



**Experiment 12**

**Write a program to Build Cat vs Dog prediction model using transfer learning**

*Code:*

!mkdir -p ~/.kaggle

!cp kaggle.json ~/.kaggle/

!kaggle datasets download -d salader/dogs-vs-cats

import zipfile

zip\_ref = zipfile.ZipFile('/content/dogs-vs-cats.zip', 'r')

zip\_ref.extractall('/content')

zip\_ref.close()

import tensorflow as tf

import keras

from keras import Sequential

from keras.layers import Dense, Flatten

from keras.applications.vgg16 import VGG16

conv\_base = VGG16(

weights = 'imagenet',

include\_top = False,

input\_shape = (150, 150, 3)

)

conv\_base.summary()

model=Sequential()

model.add(conv\_base)

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dense(1,activation='sigmoid'))

model.summary()

conv\_base.trainable = False

model.summary()

# generators

train\_ds = keras.utils.image\_dataset\_from\_directory(

directory = '/content/train',

labels = 'inferred',

label\_mode = 'int',

batch\_size=32,

image\_size=(150,150)

)

validation\_ds = keras.utils.image\_dataset\_from\_directory(

directory = '/content/test',

labels = 'inferred',

label\_mode = 'int',

batch\_size=32,

image\_size=(150,150)

)

#Normalize

def process(image,label):

image = tf.cast(image/255, tf.float32)

return image, label

train\_ds = train\_ds.map(process)

validation\_ds = validation\_ds.map(process)

model.compile(optimizer='adam',loss='binary\_crossentropy',metrics=['accuracy'])

history=model.fit(train\_ds,epochs=10, validation\_data=validation\_ds)

import matplotlib.pyplot as plt

import cv2

import cv2 as cv

import numpy as np

import argparse

import time

import cv2

test\_img = cv2.imread('/content/Dog.png')

plt.imshow(test\_img)

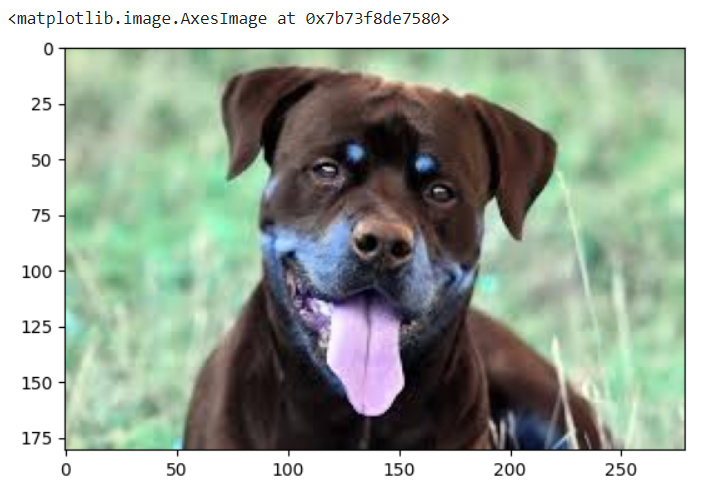
Test\_img.shape

test\_img = cv2.resize(test\_img, (150, 150))

test\_input = test\_img.reshape((1,150,150,3))

model.predict(test\_input)

*Output:*





**Experiment 13**

**Write a program to Build Long Short Term Memory**

*Code:*

faqs = """About the Program

What is the course fee for Data Science Mentorship Program (DSMP 2023)

The course follows a monthly subscription model where you have to make monthly payments of Rs 799/month.

What is the total duration of the course?

The total duration of the course is 7 months. So the total course fee becomes 799\*7 = Rs 5600(approx.)

What is the syllabus of the mentorship program?

We will be covering the following modules:

Python Fundamentals

Python libraries for Data Science

Data Analysis

SQL for Data Science

Maths for Machine Learning

ML Algorithms

Practical ML

MLOPs

Case studies

You can check the detailed syllabus here - https://learnwith.campusx.in/courses/CampusX-Data-Science-Mentorship-Program-637339afe4b0615a1bbed390

Will Deep Learning and NLP be a part of this program?

No, NLP and Deep Learning both are not a part of this program’s curriculum.

What if I miss a live session? Will I get a recording of the session?

Yes all our sessions are recorded, so even if you miss a session you can go back and watch the recording.

Where can I find the class schedule?

Checkout this google sheet to see month by month time table of the course - https://docs.google.com/spreadsheets/d/16OoTax\_A6ORAeCg4emgexhqqPv3noQPYKU7RJ6ArOzk/edit?usp=sharing.

What is the time duration of all the live sessions?

Roughly, all the sessions last 2 hours.

What is the language spoken by the instructor during the sessions?

Hinglish

How will I be informed about the upcoming class?

You will get a mail from our side before every paid session once you become a paid user.

Can I do this course if I am from a non-tech background?

Yes, absolutely.

I am late, can I join the program in the middle?

Absolutely, you can join the program anytime.

If I join/pay in the middle, will I be able to see all the past lectures?

Yes, once you make the payment you will be able to see all the past content in your dashboard.

Where do I have to submit the task?

You don’t have to submit the task. We will provide you with the solutions, you have to self evaluate the task yourself.

Will we do case studies in the program?

Yes.

Where can we contact you?

You can mail us at nitish.campusx@gmail.com

Payment/Registration related questions

Where do we have to make our payments? Your YouTube channel or website?

You have to make all your monthly payments on our website. Here is the link for our website - https://learnwith.campusx.in/

Can we pay the entire amount of Rs 5600 all at once?

Unfortunately no, the program follows a monthly subscription model.

What is the validity of monthly subscription? Suppose if I pay on 15th Jan, then do I have to pay again on 1st Feb or 15th Feb

15th Feb. The validity period is 30 days from the day you make the payment. So essentially you can join anytime you don’t have to wait for a month to end.

What if I don’t like the course after making the payment. What is the refund policy?

You get a 7 days refund period from the day you have made the payment.

I am living outside India and I am not able to make the payment on the website, what should I do?

You have to contact us by sending a mail at nitish.campusx@gmail.com

Post registration queries

Till when can I view the paid videos on the website?

This one is tricky, so read carefully. You can watch the videos till your subscription is valid. Suppose you have purchased subscription on 21st Jan, you will be able to watch all the past paid sessions in the period of 21st Jan to 20th Feb. But after 21st Feb you will have to purchase the subscription again.

But once the course is over and you have paid us Rs 5600(or 7 installments of Rs 799) you will be able to watch the paid sessions till Aug 2024.

Why lifetime validity is not provided?

Because of the low course fee.

Where can I reach out in case of a doubt after the session?

You will have to fill a google form provided in your dashboard and our team will contact you for a 1 on 1 doubt clearance session

If I join the program late, can I still ask past week doubts?

Yes, just select past week doubt in the doubt clearance google form.

I am living outside India and I am not able to make the payment on the website, what should I do?

You have to contact us by sending a mail at nitish.campusx@gmai.com

Certificate and Placement Assistance related queries

What is the criteria to get the certificate?

There are 2 criterias:

You have to pay the entire fee of Rs 5600

You have to attempt all the course assessments.

I am joining late. How can I pay payment of the earlier months?

You will get a link to pay fee of earlier months in your dashboard once you pay for the current month.

I have read that Placement assistance is a part of this program. What comes under Placement assistance?

This is to clarify that Placement assistance does not mean Placement guarantee. So we dont guarantee you any jobs or for that matter even interview calls. So if you are planning to join this course just for placements, I am afraid you will be disappointed. Here is what comes under placement assistance

Portfolio Building sessions

Soft skill sessions

Sessions with industry mentors

Discussion on Job hunting strategies

"""

import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts([faqs])

len(tokenizer.word\_index)

input\_sequences = []

for sentence in faqs.split('\n'):

tokenized\_sentence = tokenizer.texts\_to\_sequences([sentence])[0]

for i in range(1,len(tokenized\_sentence)):

input\_sequences.append(tokenized\_sentence[:i+1])

print(input\_sequences)

max\_len = max([len(x) for x in input\_sequences])

print(max\_len)

from tensorflow.keras.preprocessing.sequence import pad\_sequences

padded\_input\_sequences = pad\_sequences(input\_sequences, maxlen = max\_len, padding='pre')

print(padded\_input\_sequences)

x = padded\_input\_sequences[:,:-1]

print(x)

y = padded\_input\_sequences[:,-1]

print(y)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense

model = Sequential()

model.add(Embedding(283, 100, input\_length = 56))

model.add(LSTM(150))

model.add(Dense(283, activation='softmax'))

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()

from tensorflow.keras.utils import to\_categorical

y = to\_categorical(y, num\_classes = 283)

print(y.shape)



model.fit(x,y,epochs = 100)

import numpy as np

import time

text = "No, NLP and Deep"

for i in range(10):

#tokenize

token\_text = tokenizer.texts\_to\_sequences([text])[0]

#padding

padded\_token\_text = pad\_sequences([token\_text],maxlen=56, padding='pre')

#predict

pos = np.argmax(model.predict(padded\_token\_text))

for word,index in tokenizer.word\_index.items():

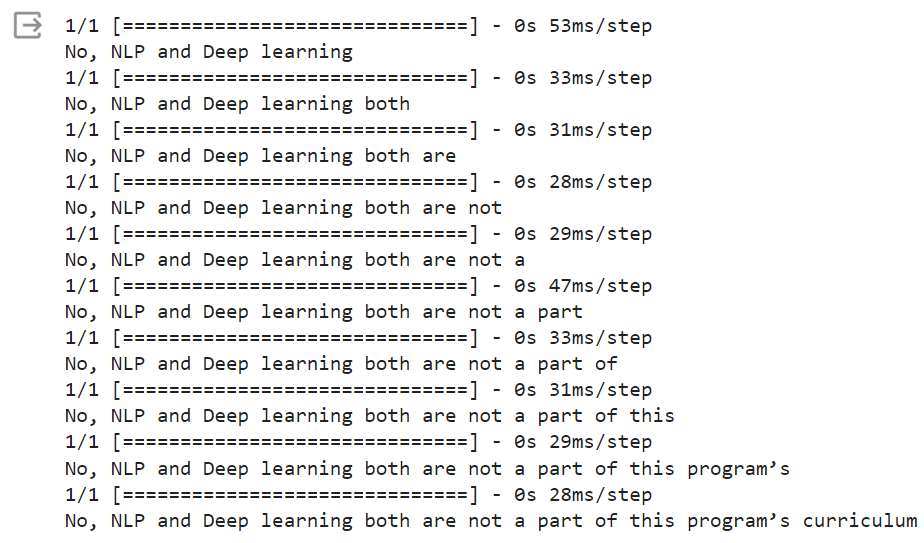
if index == pos:

text = text + " " + word

print(text)

time.sleep(2)

*Output:*



**Experiment 14**

**Write a program Integer Encoding Using Simple RNN.**

*Code:*

import numpy as np

docs = [

"go india",

"india india","hip hip hurray","jeetega bhai jeetega india jeetega","bharat mata ki jai","kholi kholi",

"sachin sachin","dhoni dhoni","modi ji ki jai","inqualab jindabad"

]

from keras.preprocessing.text import Tokenizer

tokenizer= Tokenizer(oov\_token="<nothing>")

tokenizer.fit\_on\_texts(docs)

tokenizer.word\_index

tokenizer.word\_counts

tokenizer.document\_count

sequences = tokenizer.texts\_to\_sequences(docs)

sequences

from keras.utils import pad\_sequences

sequences = pad\_sequences(sequences, padding='post')

sequences

from keras.datasets import imdb

from keras import Sequential

from keras.layers import Dense, SimpleRNN, Embedding, Flatten

(x\_train, y\_train),(x\_test,y\_test) = imdb.load\_data()

(x\_train, y\_train),(x\_test,y\_test) = imdb.load\_data()

len(x\_train[2])

x\_train = pad\_sequences(x\_train,padding='post', maxlen=50)

x\_test = pad\_sequences(x\_test,padding='post', maxlen=50)

x\_train[0]

model = Sequential()

model.add(SimpleRNN(32, input\_shape=(50,1),return\_sequences=False))

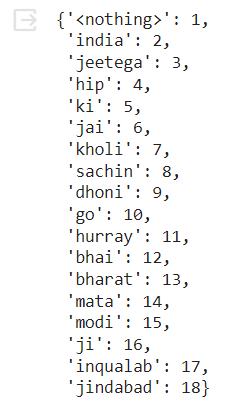
model.add(Dense(1,activation='sigmoid'))

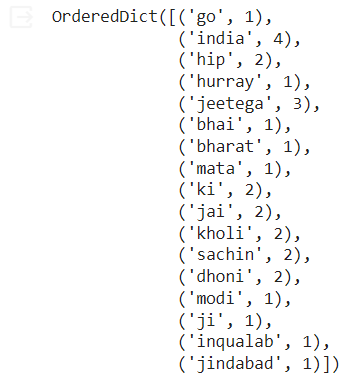
model.summary()

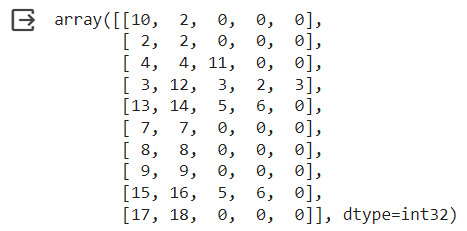
model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

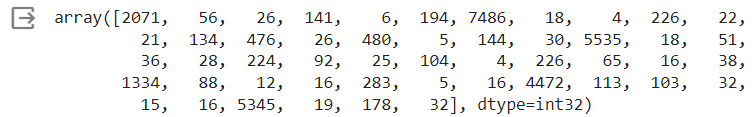
model.fit(x\_train,y\_train,epochs=5,validation\_data=(x\_test,y\_test))

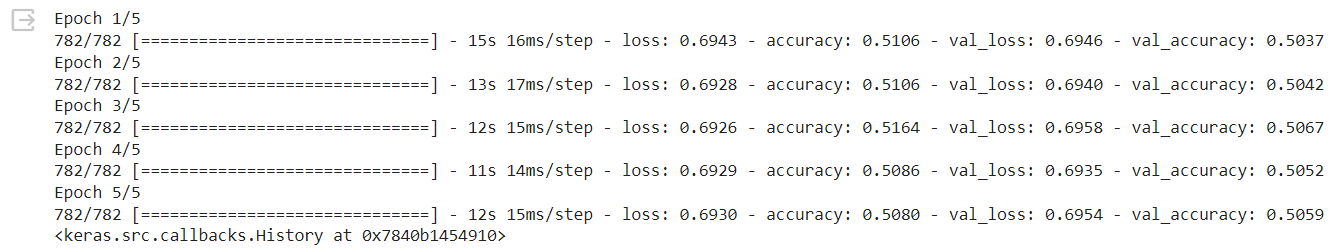
*Output:*











**Experiment 15**

**Write a program to Build Embeading Sentiment Analysis Using Simple RNN.**

*Code:*

docs = [

"go india",

"india india","hip hip hurray","jeetega bhai jeetega india jeetega","bharat mata ki jai","kholi kholi",

"sachin sachin","dhoni dhoni","modi ji ki jai","inqualab jindabad"

]

from keras.datasets import imdb

from keras import Sequential

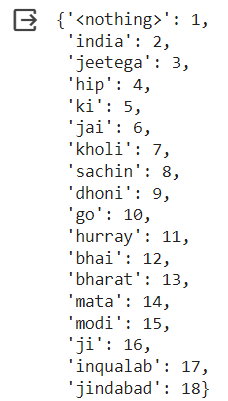
from keras.layers import Dense, SimpleRNN, Embedding, Flatten

from keras.preprocessing.text import Tokenizer

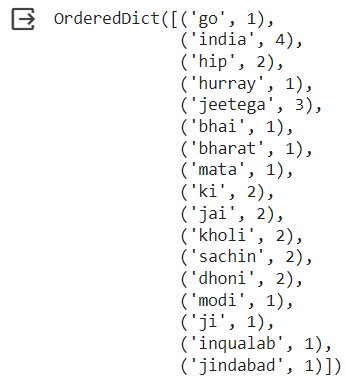
tokenizer= Tokenizer(oov\_token="<nothing>")

tokenizer.fit\_on\_texts(docs)

tokenizer.word\_index



tokenizer.word\_counts

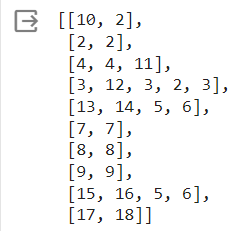


tokenizer.document\_count



sequences = tokenizer.texts\_to\_sequences(docs)

sequences



(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data()

from keras.utils import pad\_sequences

x\_train = pad\_sequences(x\_train, padding='post',maxlen=50)

x\_test = pad\_sequences(x\_test,padding='post', maxlen=50)

x\_train.shape



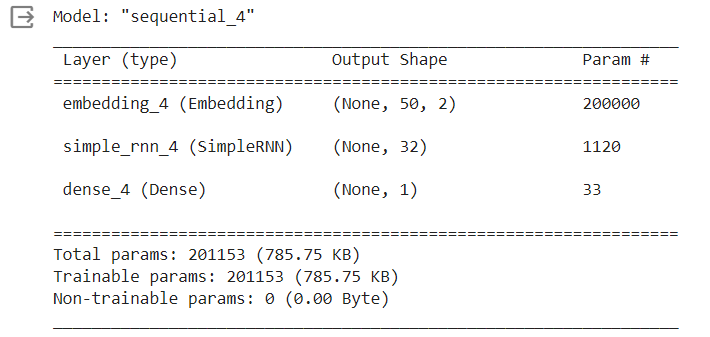
model = Sequential()

model.add(Embedding(100000, output\_dim=2, input\_length=50))

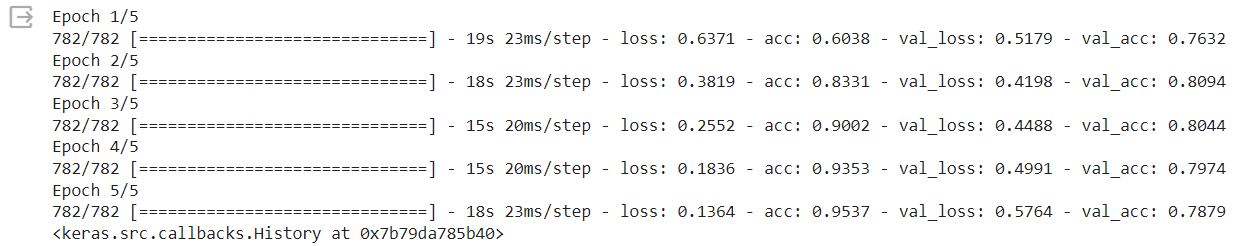
model.add(SimpleRNN(32, return\_sequences=False))

model.add(Dense(1,activation='sigmoid'))

model.summary()



model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['acc'])



**Experiment 16**

**Write a program for Logistic regression model (Spam-ham)**

*Code:*

# Reading Data

import pandas as pd

data = pd.read\_csv('https://raw.githubusercontent.com/mohitgupta-omg/Kaggle-SMS-Spam-Collection-Dataset-/master/spam.csv', encoding='latin-1')

data.head()

data.drop(['Unnamed: 2','Unnamed: 3','Unnamed: 4'],axis=1, inplace=True)

data.columns = ['label', 'text']

data.head()

data.isna().sum()

import nltk

nltk.download('all')

text = list(data['text'])

import re

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

lematizer= WordNetLemmatizer()

corpus = []

for i in range(len(text)):

r = re.sub('[^a=zA-Z]',' ',text[i])

r = r.lower()

r = r.split()

r = [word for word in r if word not in stopwords.words('english')]

r = ' '.join(r)

corpus.append(r)

data['text'] = corpus

data.head()

X=data['text']

y=data['label']

from sklearn.model\_selection import train\_test\_split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.33,random\_state=123)

print("Training Data:",X\_train.shape)

print("Testing Data:",X\_test.shape)

from sklearn.feature\_extraction.text import CountVectorizer

cv=CountVectorizer()

X\_train\_cv=cv.fit\_transform(X\_train)

X\_train\_cv.shape

#Training Logistic Regressio Model

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X\_train\_cv, y\_train)

X\_test\_cv = cv.transform(X\_test)  
predictions = lr.predict(X\_test\_cv)

predictions

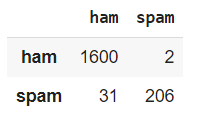
import pandas as pd

from sklearn import metrics

df = pd.DataFrame(metrics.confusion\_matrix(y\_test,predictions), index=['ham','spam'], columns=['ham','spam'])

df

*Output:*



**Experiment 17**

**Write a program for ANN classification.**

# Reading the cleaned numeric titanic survival data

import pandas as pd

import numpy as np

# To remove the scientific notation from numpy arrays

np.set\_printoptions(suppress=True)

TitanicSurvivalDataNumeric=pd.read\_pickle('TitanicSurvivalDataNumeric.pkl')

TitanicSurvivalDataNumeric.head()

# Separate Target Variable and Predictor Variables

TargetVariable=['Survived']

Predictors=['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare',

'Embarked\_C', 'Embarked\_Q', 'Embarked\_S']

X=TitanicSurvivalDataNumeric[Predictors].values

y=TitanicSurvivalDataNumeric[TargetVariable].values

### Sandardization of data ###

### We does not standardize the Target variable for classification

from sklearn.preprocessing import StandardScaler

PredictorScaler=StandardScaler()

# Storing the fit object for later reference

PredictorScalerFit=PredictorScaler.fit(X)

# Generating the standardized values of X and y

X=PredictorScalerFit.transform(X)

# Split the data into training and testing set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Quick sanity check with the shapes of Training and Testing datasets

print(X\_train.shape)

print(y\_train.shape)

print(X\_test.shape)

print(y\_test.shape)

#Build Artificial Neural Network

#Import the Keras libraries and packages

import keras

from keras.models import Sequential

from keras.layers import Dense

classifier = Sequential()

# Defining the Input layer and FIRST hidden layer,both are same!

# relu means Rectifier linear unit function

classifier.add(Dense(units=10, input\_dim=9, kernel\_initializer='uniform', activation='relu'))

#Defining the SECOND hidden layer, here we have not defined input because it is

# second layer and it will get input as the output of first hidden layer

classifier.add(Dense(units=6, kernel\_initializer='uniform', activation='relu'))

# Defining the Output layer

# sigmoid means sigmoid activation function

# for Multiclass classification the activation ='softmax'

# And output\_dim will be equal to the number of factor levels

classifier.add(Dense(units=1, kernel\_initializer='uniform', activation='sigmoid'))

# Optimizer== the algorithm of SGG to keep updating weights

# loss== the loss function to measure the accuracy

# metrics== the way we will compare the accuracy after each step of SGD

classifier.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# fitting the Neural Network on the training data

survivalANN\_Model=classifier.fit(X\_train,y\_train, batch\_size=10 , epochs=10, verbose=1)

# fitting the Neural Network on the training data

survivalANN\_Model=classifier.fit(X\_train,y\_train, batch\_size=10 , epochs=10, verbose=1)

# Defining a function for finding best hyperparameters

def FunctionFindBestParams(X\_train, y\_train):

# Defining the list of hyper parameters to try

TrialNumber=0

batch\_size\_list=[5, 10, 15, 20]

epoch\_list=[5, 10, 50 ,100]

import pandas as pd

SearchResultsData=pd.DataFrame(columns=['TrialNumber', 'Parameters', 'Accuracy'])

for batch\_size\_trial in batch\_size\_list:

for epochs\_trial in epoch\_list:

TrialNumber+=1

# Creating the classifier ANN model

classifier = Sequential()

classifier.add(Dense(units=10, input\_dim=9, kernel\_initializer='uniform', activation='relu'))

classifier.add(Dense(units=6, kernel\_initializer='uniform', activation='relu'))

classifier.add(Dense(units=1, kernel\_initializer='uniform', activation='sigmoid'))

classifier.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

survivalANN\_Model=classifier.fit(X\_train,y\_train, batch\_size=batch\_size\_trial , epochs=epochs\_trial, verbose=0)

# Fetching the accuracy of the training

Accuracy = survivalANN\_Model.history['accuracy'][-1]

# printing the results of the current iteration

print(TrialNumber, 'Parameters:','batch\_size:', batch\_size\_trial,'-', 'epochs:',epochs\_trial, 'Accuracy:', Accuracy)

SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumber,

'batch\_size'+str(batch\_size\_trial)+'-'+'epoch'+str(epochs\_trial), Accuracy]],

columns=['TrialNumber', 'Parameters', 'Accuracy'] ))

return(SearchResultsData)

# Calling the function

ResultsData=FunctionFindBestParams(X\_train, y\_train)

# Printing the best parameter

print(ResultsData.sort\_values(by='Accuracy', ascending=False).head(1))

# Visualizing the results

%matplotlib inline

ResultsData.plot(x='Parameters', y='Accuracy', figsize=(15,4), kind='line', rot=20)

# Training the model with best hyperparamters

classifier.fit(X\_train,y\_train, batch\_size=5 , epochs=100, verbose=1)

# Predictions on testing data

Predictions=classifier.predict(X\_test)

# Scaling the test data back to original scale

Test\_Data=PredictorScalerFit.inverse\_transform(X\_test)

# Generating a data frame for analyzing the test data

TestingData=pd.DataFrame(data=Test\_Data, columns=Predictors)

TestingData['Survival']=y\_test

TestingData['PredictedSurvivalProb']=Predictions

# Defining the probability threshold

def probThreshold(inpProb):

if inpProb > 0.5:

return(1)

else:

return(0)

# Generating predictions on the testing data by applying probability threshold

TestingData['PredictedSurvival']=TestingData['PredictedSurvivalProb'].apply(probThreshold)

print(TestingData.head())

from sklearn import metrics

print('\n######### Testing Accuracy Results #########')

print(metrics.classification\_report(TestingData['Survival'], TestingData['PredictedSurvival']))

print(metrics.confusion\_matrix(TestingData['Survival'], TestingData['PredictedSurvival']))

# Function to generate Deep ANN model

def make\_classification\_ann(Optimizer\_Trial, Neurons\_Trial):

from keras.models import Sequential

from keras.layers import Dense

# Creating the classifier ANN model

classifier = Sequential()

classifier.add(Dense(units=Neurons\_Trial, input\_dim=9, kernel\_initializer='uniform', activation='relu'))

classifier.add(Dense(units=Neurons\_Trial, kernel\_initializer='uniform', activation='relu'))

classifier.add(Dense(units=1, kernel\_initializer='uniform', activation='sigmoid'))

classifier.compile(optimizer=Optimizer\_Trial, loss='binary\_crossentropy', metrics=['accuracy'])

return classifier

from sklearn.model\_selection import GridSearchCV

from keras.wrappers.scikit\_learn import KerasClassifier

Parameter\_Trials={'batch\_size':[10,20,30],

'epochs':[10,20],

'Optimizer\_Trial':['adam', 'rmsprop'],

'Neurons\_Trial': [5,10]

}

# Creating the classifier ANN

classifierModel=KerasClassifier(make\_classification\_ann, verbose=0)

# Creating the Grid search space

# See different scoring methods by using sklearn.metrics.SCORERS.keys()

grid\_search=GridSearchCV(estimator=classifierModel, param\_grid=Parameter\_Trials, scoring='f1', cv=5)

# Measuring how much time it took to find the best params

import time

StartTime=time.time()

# Running Grid Search for different paramenters

grid\_search.fit(X\_train,y\_train, verbose=1)

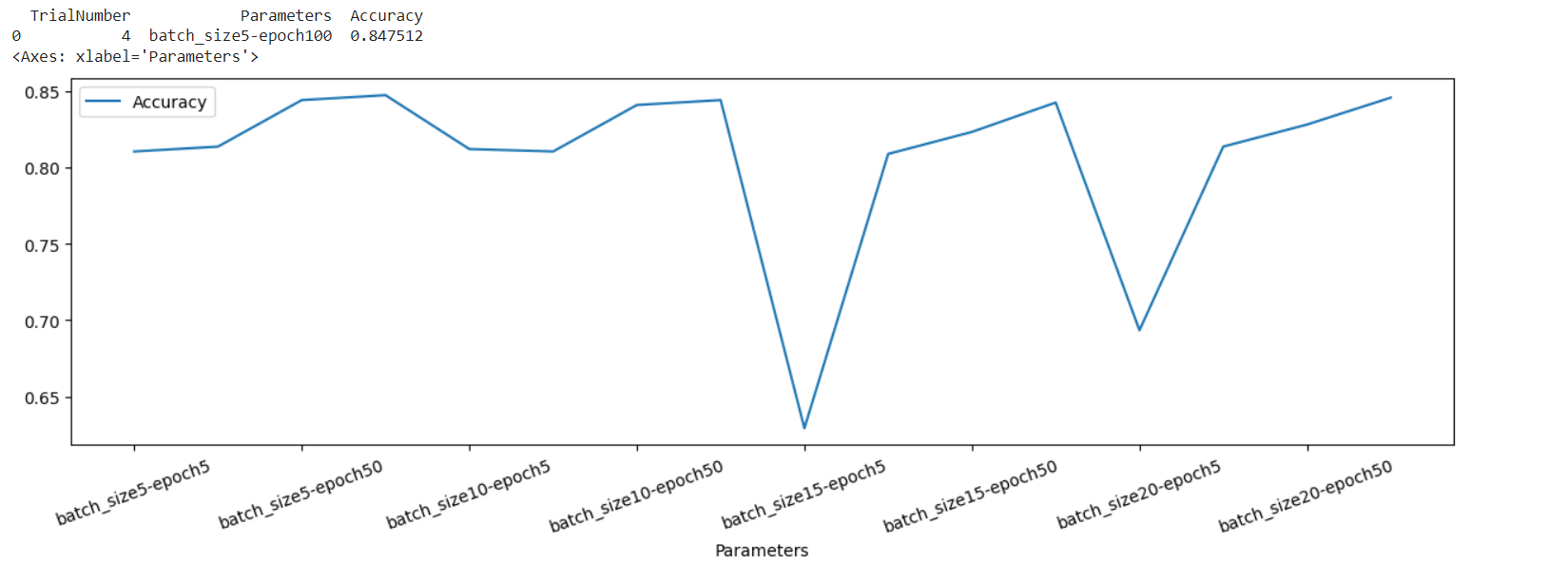
EndTime=time.time()

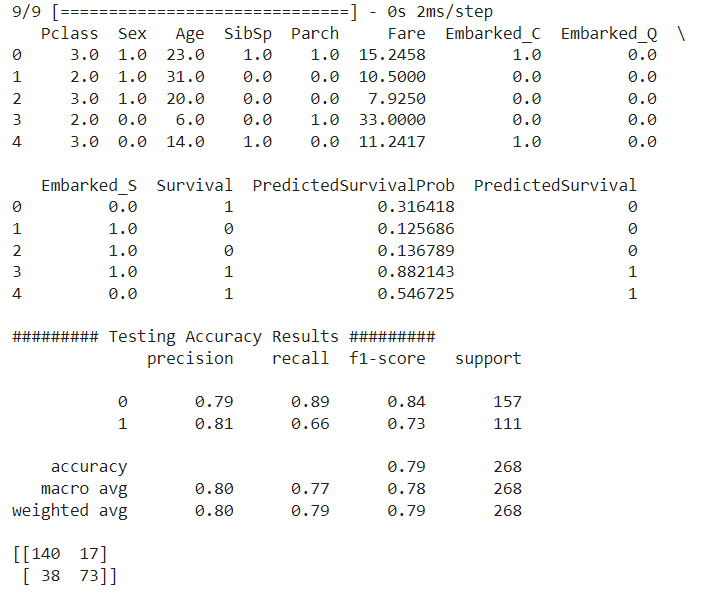
print("############### Total Time Taken: ", round((EndTime-StartTime)/60), 'Minutes #############')

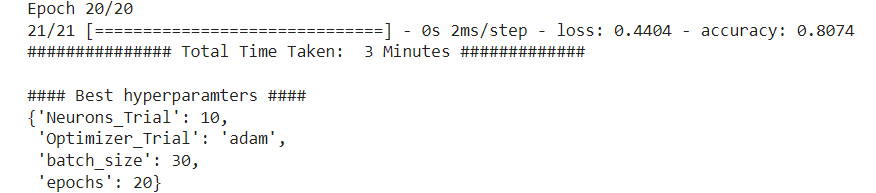
# printing the best parameters

print('\n#### Best hyperparamters ####')

grid\_search.best\_params\_

 *Output:*





**Experiment 18**

**Write a program for ANN regression.**

from google.colab import files

uploaded=files.upload()

# Reading the cleaned numeric car prices data

import pandas as pd

import numpy as np

# To remove the scientific notation from numpy arrays

np.set\_printoptions(suppress=True)

CarPricesDataNumeric=pd.read\_pickle('CarPricesData.pkl')

CarPricesDataNumeric.head()

# Separate Target Variable and Predictor Variables

TargetVariable=['Price']

Predictors=['Age', 'KM', 'Weight', 'HP', 'MetColor', 'CC', 'Doors']

X=CarPricesDataNumeric[Predictors].values

y=CarPricesDataNumeric[TargetVariable].values

### Sandardization of data ###

from sklearn.preprocessing import StandardScaler

PredictorScaler=StandardScaler()

TargetVarScaler=StandardScaler()

# Storing the fit object for later reference

PredictorScalerFit=PredictorScaler.fit(X)

TargetVarScalerFit=TargetVarScaler.fit(y)

# Generating the standardized values of X and y

X=PredictorScalerFit.transform(X)

y=TargetVarScalerFit.transform(y)

# Split the data into training and testing set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Quick sanity check with the shapes of Training and testing datasets

print(X\_train.shape)

print(y\_train.shape)

print(X\_test.shape)

print(y\_test.shape)

# importing the libraries

from keras.models import Sequential

from keras.layers import Dense

# create ANN model

model = Sequential()

# Defining the Input layer and FIRST hidden layer, both are same!The term kernel\_initializer is a fancy term

# for which statistical distribution or function to use for initialising the weights

model.add(Dense(units=5, input\_dim=7, kernel\_initializer='normal', activation='relu'))

# Defining the Second layer of the model

# after the first layer we don't have to specify input\_dim as keras configure it automatically

model.add(Dense(units=5, kernel\_initializer='normal', activation='tanh'))

# The output neuron is a single fully connected node

# Since we will be predicting a single number

model.add(Dense(1, kernel\_initializer='normal'))

# Compiling the model

#Adam Optimizer is a technique that reduces the time taken to train a model in Deep Learning.

model.compile(loss='mean\_squared\_error', optimizer='adam')

# Fitting the ANN to the Training set

#verbose = 1, which includes both progress bar and one line per epoch

#verbose = 0, means silent

# one line per epoch i.e. epoch no./total no. of epochs

model.fit(X\_train, y\_train ,batch\_size = 20, epochs = 50, verbose=1)

model.summary()

# Defining a function to find the best parameters for ANN

def FunctionFindBestParams(X\_train, y\_train, X\_test, y\_test):

# Defining the list of hyper parameters to try

batch\_size\_list=[5, 10, 15, 20]

epoch\_list = [5, 10, 50, 100]

import pandas as pd

SearchResultsData=pd.DataFrame(columns=['TrialNumber', 'Parameters', 'Accuracy'])

# initializing the trials

TrialNumber=0

for batch\_size\_trial in batch\_size\_list:

for epochs\_trial in epoch\_list:

TrialNumber+=1

# create ANN model

model = Sequential()

# Defining the first layer of the model

model.add(Dense(units=5, input\_dim=X\_train.shape[1], kernel\_initializer='normal', activation='relu'))

# Defining the Second layer of the model

model.add(Dense(units=5, kernel\_initializer='normal', activation='relu'))

# The output neuron is a single fully connected node

# Since we will be predicting a single number

model.add(Dense(1, kernel\_initializer='normal'))

# Compiling the model

model.compile(loss='mean\_squared\_error', optimizer='adam')

# Fitting the ANN to the Training set

model.fit(X\_train, y\_train ,batch\_size = batch\_size\_trial, epochs = epochs\_trial, verbose=0)

MAPE = np.mean(100 \* (np.abs(y\_test-model.predict(X\_test))/y\_test))

# printing the results of the current iteration

print(TrialNumber, 'Parameters:','batch\_size:', batch\_size\_trial,'-', 'epochs:',epochs\_trial, 'Accuracy:', 100-MAPE)

SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumber, str(batch\_size\_trial)+'-'+str(epochs\_trial), 100-MAPE]],

columns=['TrialNumber', 'Parameters', 'Accuracy'] ))

return(SearchResultsData)

######################################################

# Calling the function

ResultsData=FunctionFindBestParams(X\_train, y\_train, X\_test, y\_test)

%matplotlib inline

ResultsData.plot(x='Parameters', y='Accuracy', figsize=(15,4), kind='line')

# Fitting the ANN to the Training set

model.fit(X\_train, y\_train ,batch\_size = 15, epochs = 5, verbose=0)

# Generating Predictions on testing data

Predictions=model.predict(X\_test)

# Scaling the predicted Price data back to original price scale

Predictions=TargetVarScalerFit.inverse\_transform(Predictions)

# Scaling the y\_test Price data back to original price scale

y\_test\_orig=TargetVarScalerFit.inverse\_transform(y\_test)

# Scaling the test data back to original scale

Test\_Data=PredictorScalerFit.inverse\_transform(X\_test)

TestingData=pd.DataFrame(data=Test\_Data, columns=Predictors)

TestingData['Price']=y\_test\_orig

TestingData['PredictedPrice']=Predictions

TestingData.head()

# Computing the absolute percent error

APE=100\*(abs(TestingData['Price']-TestingData['PredictedPrice'])/TestingData['Price'])

TestingData['APE']=APE

print('The Accuracy of ANN model is:', 100-np.mean(APE))

TestingData.head()

# Function to generate Deep ANN model

def make\_regression\_ann(Optimizer\_trial):

from keras.models import Sequential

from keras.layers import Dense

model = Sequential()

model.add(Dense(units=5, input\_dim=7, kernel\_initializer='normal', activation='relu'))

model.add(Dense(units=5, kernel\_initializer='normal', activation='relu'))

model.add(Dense(1, kernel\_initializer='normal'))

model.compile(loss='mean\_squared\_error', optimizer=Optimizer\_trial)

return model

###########################################

from sklearn.model\_selection import GridSearchCV

from keras.wrappers.scikit\_learn import KerasRegressor

# Listing all the parameters to try

Parameter\_Trials={'batch\_size':[10,20,30],

'epochs':[10,20],

'Optimizer\_trial':['adam', 'rmsprop']

}

# Creating the regression ANN model

RegModel=KerasRegressor(make\_regression\_ann, verbose=0)

###########################################

from sklearn.metrics import make\_scorer

# Defining a custom function to calculate accuracy

def Accuracy\_Score(orig,pred):

MAPE = np.mean(100 \* (np.abs(orig-pred)/orig))

print('#'\*70,'Accuracy:', 100-MAPE)

return(100-MAPE)

custom\_Scoring=make\_scorer(Accuracy\_Score, greater\_is\_better=True)

#########################################

# Creating the Grid search space

# See different scoring methods by using sklearn.metrics.SCORERS.keys()

grid\_search=GridSearchCV(estimator=RegModel,

param\_grid=Parameter\_Trials,

scoring=custom\_Scoring,

cv=5)

#########################################

# Measuring how much time it took to find the best params

import time

StartTime=time.time()

# Running Grid Search for different paramenters

grid\_search.fit(X,y, verbose=1)

EndTime=time.time()

print("########## Total Time Taken: ", round((EndTime-StartTime)/60), 'Minutes')

print('### Printing Best parameters ###')

grid\_search.best\_params\_