410255: LP-V. DL 2.B 22-23 (2^{nd} Sem)					
Student Name :					
BE Computer Division		Roll No		Batch	
Experiment No: 2B	Classification using Deep neural network – **Binary Classification**				

Aim : Classification using Deep neural network : (Any One from the following)

1. Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition dataset https://archive.ics.uci.edu/ml/datasets/letter+recognition
2. Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset

2. Binary Classification using Deep Learning

Objective of the Assignment: Students should be able to Classify movie reviews into positive reviews
and "negative reviews on IMDB Dataset.

Prerequisite:

1. Basic of programming language
2. Concept of Classification
3. Concept of Deep Neural Network

Contents for Theory:

1. What is Classification
2. Example of Classification
3. How Deep Neural Network Work on Classification
4. Code Explanation with Output

What is Classification?

Classification is a type of supervised learning in machine learning that involves categorizing data into predefined classes or categories based on a set of features or characteristics. It is used to predict the class of new, unseen data based on the patterns learned from the labeled training data.

In classification, a model is trained on a labeled dataset, where each data point has a known class label. The model learns to associate the input features with the corresponding class labels and can then be used to classify new, unseen data.

For example, we can use classification to identify whether an email is spam or not based on its content and metadata, to predict whether a patient has a disease

based on their medical records and symptoms, or to classify images into different categories based on their visual features.
Classification algorithms can vary in complexity, ranging from simple models such as decision trees and k-nearest neighbours to more complex models such as support vector machines and neural networks. The choice of algorithm depends on the nature of the data, the size of the dataset, and the desired level of accuracy and interpretability.

Classification is a common task in deep neural networks, where the goal is to predict the class of an input based on its features. Here's an example of how classification can be performed in a deep neural network using the popular MNIST dataset of handwritten digits.
The MNIST dataset contains 60,000 training images and 10,000 testing images of handwritten digits from 0 to 9. Each image is a grayscale 28x28 pixel image, and the task is to classify each image into one of the 10 classes corresponding to the 10 digits.
We can use a convolutional neural network (CNN) to classify the MNIST dataset. A CNN is a type of
deep neural network that is commonly used for image classification tasks.

How Deep Neural Network Work on Classification-
Deep neural networks are commonly used for classification tasks because they can automatically learn to extract relevant features from raw input data and map them to the correct output class.
The basic architecture of a deep neural network for classification consists of three main parts: an input layer, one or more hidden layers, and an output layer. The input layer receives the raw input data, which is usually prepossessed to a fixed size and format. The hidden layers are composed of neurons that apply linear transformations and nonlinear activations to the input features to extract relevant patterns and representations. Finally, the output layer produces the predicted class labels, usually as a probability distribution over the possible classes.
During training, the deep neural network learns to adjust its weights and biases in each layer to minimize the difference between the predicted output and the true labels. This is typically done by optimizing a loss function that measures the discrepancy between the predicted and true labels, using techniques such as gradient descent or stochastic gradient descent. One of the key advantages of deep neural networks for classification is their ability to learn hierarchical
representations of the input data. In a deep neural network with multiple hidden layers, each layer learns to capture more complex and abstract features than the previous layer, by building on the representations learned by the earlier layers. This hierarchical structure allows deep neural networks to learn highly discriminative features that can separate different classes of input data, even when the data is highly complex or noisy.
Overall, the effectiveness of deep neural networks for classification depends on the choice of
architecture, hyperparameters, and training procedure, as well as the quality and quantity of the training data. When trained properly, deep neural networks can achieve state-of-the-art performance on a wide range of classification tasks, from image recognition to natural language processing.
IMDB Dataset-The IMDB dataset is a large collection of movie reviews collected from

the IMDB website, which is a popular source of user-generated movie ratings and reviews. The dataset consists of 50,000 movie reviews, split into 25,000 reviews for training and 25,000 reviews for testing.
Each review is represented as a sequence of words, where each word is represented by an integer index based on its frequency in the dataset. The labels for each review are binary, with 0 indicating a negative review and 1 indicating a positive review.
The IMDB dataset is commonly used as a benchmark for sentiment analysis and text classification tasks, where the goal is to classify the movie reviews as either positive or negative based on their text content. The dataset is challenging because the reviews are often highly subjective and can contain complex language and nuances of meaning, making it difficult for traditional machine learning approaches to accurately classify them.
Deep learning approaches, such as deep neural networks, have achieved state-of-the-art performance on the IMDB dataset by automatically learning to extract relevant features from the raw text data and map them to the correct output class. The IMDB dataset is widely used in research and education for natural language processing and machine learning, as it provides a rich source of labeled text data for training and testing deep learning models.
Conclusion- In this way we can Classify the Movie Reviews by using DNN.

Assignment Question on Binary classification

1. What is Binary Classification?
2. What is binary Cross Entropy?
3. What is Validation Split?
4. What is the Epoch Cycle?
5. What is Adam Optimizer?

Attendance (5 Marks)	Conduction (5 Marks)	Oral (5 Marks)	Total (15 Marks)	Sign of Teacher

Source Code and Output-

```
# The IMDB sentiment classification dataset consists of 50,000 movie reviews from IMDB users
that are
labeled as either positive (1) or negative (0).
# The reviews are preprocessed and each one is encoded as a sequence of word indexes in the
form of
integers.
# The words within the reviews are indexed by their overall frequency within the dataset. For
example,
the integer “2” encodes the second most frequent word in the data.
# The 50,000 reviews are split into 25,000 for training and 25,000 for testing.
# Text Process word by word at diffrent timestamp ( You may use RNN LSTM GRU )
# convert input text to vector reprent input text
# DOMAIN: Digital content and entertainment industry
# CONTEXT: The objective of this project is to build a text classification model that analyses the
customer's sentiments based on their reviews in the IMDB database. The model uses a complex
deep
learning model to build an embedding layer followed by a classification algorithm to analyse the
sentiment of the customers.
# DATA DESCRIPTION: The Dataset of 50,000 movie reviews from IMDB, labelled by
sentiment
(positive/negative).
# Reviews have been preprocessed, and each review is encoded as a sequence of word indexes
(integers).
# For convenience, the words are indexed by their frequency in the dataset, meaning the for that
has
index 1 is the most frequent word.
# Use the first 20 words from each review to speed up training, using a max vocabulary size of
10,000.
# As a convention, "0" does not stand for a specific word, but instead is used to encode any
unknown
word.
# PROJECT OBJECTIVE: Build a sequential NLP classifier which can use input text parameters
to
determine the customer sentiments.
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
#loading imdb data with most frequent 10000 words
from keras.datasets import imdb
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000) # you may take top
10,000
word frequently used review of movies other are discarded
#consolidating data for EDA Exploratory data analysis (EDA) is used by data scientists to
```

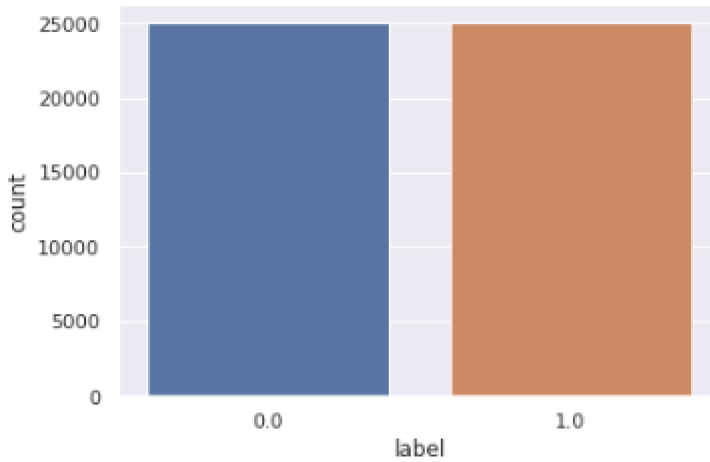
analyze and
investigate data sets and summarize their main characteristics

```
data = np.concatenate((X_train, X_test), axis=0) # axis 0 is first running vertically downwards across
rows (axis 0), axis 1 is second running horizontally across columns (axis 1),
label = np.concatenate((y_train, y_test), axis=0)
X_train.shape
(25000,)
X_test.shape
(25000,)
y_train.shape
(25000,)
y_test.shape
(25000,)
print("Review is ",X_train[0]) # series of no converted word to vocabulory associated with index
print("Review is ",y_train[0])
Review is [1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 118, 1634, 14,
394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114,
9, 2300, 1523, 5, 647, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5,
89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11, 3215, 2, 4,
1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15, 123, 125, 68, 2, 6853, 15, 349, 165,
4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 175, 136, 50, 9, 4373, 228, 8255,
5, 2, 656, 245, 2350, 5, 4, 9837, 131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625, 64,
1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145, 95]
Review is 0
vocab=imdb.get_word_index() # Retrieve the word index file mapping words to indices
print(vocab)
{'fawn': 34701, 'tsukino': 52006, 'nunnery': 52007, 'sonja': 16816, 'vani': 63951, 'woods': 1408, 'spiders':
16115,
y_train
array([1, 0, 0, ..., 0, 1, 0])
y_test
array([0, 1, 1, ..., 0, 0, 0])
# Function to perform relevant sequence adding on the data
# Now it is time to prepare our data. We will vectorize every review and fill it with zeros so that it
contains exactly 10000 numbers.
```

```
# That means we fill every review that is shorter than 500 with zeros.
# We do this because the biggest review is nearly that long and every input for our neural
network needs
to have the same size.
# We also transform the targets into floats.
# sequences is name of method the review less than 10000 we perform padding overthere
# binary vectorization code:
# VECTORIZE as one cannot feed integers into a NN
# Encoding the integer sequences into a binary matrix - one hot encoder basically
# From integers representing words, at various lengths - to a normalized one hot encoded tensor
(matrix)
of 10k columns
def vectorize(sequences, dimension = 10000): # We will vectorize every review and fill it with
zeros
so that it contains exactly 10,000 numbers.
# Create an all-zero matrix of shape (len(sequences), dimension)
results = np.zeros((len(sequences), dimension))
for i, sequence in enumerate(sequences):
results[i, sequence] = 1
return results
# Now we split our data into a training and a testing set.
# The training set will contain reviews and the testing set
# # Set a VALIDATION set
test_x = data[:10000]
test_y = label[:10000]
train_x = data[10000:]
train_y = label[10000:]
test_x.shape
(10000,)
test_y.shape
(10000,)
train_x.shape
(40000,)
train_y.shape
(40000,)
print("Categories:", np.unique(label))
print("Number of unique words:", len(np.unique(np.hstack(data))))
# The hstack() function is used to stack arrays in sequence horizontally (column wise).
Categories: [0 1]
Number of unique words: 9998
length = [len(i) for i in data]
print("Average Review length:", np.mean(length))
print("Standard Deviation:", round(np.std(length)))
# The whole dataset contains 9998 unique words and the average review length is 234 words,
with a
standard deviation of 173 words.
```

```
Average Review length: 234.75892
Standard Deviation: 173
# If you look at the data you will realize it has been already pre-processed.
# All words have been mapped to integers and the integers represent the words sorted by their
frequency.
# This is very common in text analysis to represent a dataset like this.
# So 4 represents the 4th most used word,
# 5 the 5th most used word and so on...
# The integer 1 is reserved for the start marker,
# the integer 2 for an unknown word and 0 for padding.
# Let's look at a single training example:
print("Label:", label[0])
Label: 1
print("Label:", label[1])
Label: 0
print(data[0])
# Retrieves a dict mapping words to their index in the IMDB dataset.
index = imdb.get_word_index() # word to index
# Create inverted index from a dictionary with document ids as keys and a list of terms as values
for
each document
reverse_index = dict([(value, key) for (key, value) in index.items()]) # id to word
decoded = " ".join( [reverse_index.get(i - 3, "#") for i in data[0]] )
# The indices are offset by 3 because 0, 1 and 2 are reserved indices for "padding", "start of
sequence"
and "unknown".
print(decoded)
# this film was just brilliant casting location scenery story direction everyone's really suited the
part they
played and you could just imagine being there robert # is an amazing actor and now the same
being
director # father came from the same scottish island as myself so i loved the fact there was a real
connection with this film the witty remarks throughout the film
#Adding sequence to data
# Vectorization is the process of converting textual data into numerical vectors and is a process
that is
usually applied once the text is cleaned.
data = vectorize(data)
label = np.array(label).astype("float32")
labelDF=pd.DataFrame({'label':label})
sns.countplot(x='label', data=labelDF)
```

```
<AxesSubplot:xlabel='label', ylabel='count'>
```




```
# Creating train and test data set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data,label, test_size=0.20, random_state=1)
X_train.shape
(40000, 10000)
X_test.shape
(10000, 10000)
# Let's create sequential model
from keras.utils import to_categorical
from keras import models
from keras import layers
model = models.Sequential()
# Input - Layer
# Note that we set the input-shape to 10,000 at the input-layer because our reviews are 10,000
integers
long.
# The input-layer takes 10,000 as input and outputs it with a shape of 50.
model.add(layers.Dense(50, activation = "relu", input_shape=(10000, )))
# Hidden - Layers
# Please note you should always use a dropout rate between 20% and 50%. # here in our case 0.3
means
30% dropout we are using dropout to prevent overfitting.
# By the way, if you want you can build a sentiment analysis without LSTMs, then you simply
need to
replace it by a flatten layer:
model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
model.add(layers.Dropout(0.2, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
# Output- Layer
model.add(layers.Dense(1, activation = "sigmoid"))
model.summary()
Model: "sequential"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 50)                500050
dropout (Dropout)            (None, 50)                0
dense_1 (Dense)              (None, 50)                2550
dropout_1 (Dropout)          (None, 50)                 0
dense_2 (Dense)              (None, 50)                 2550
dense_3 (Dense)              (None, 1)                 51
=================================================================
Total params: 505,201
Trainable params: 505,201
Non-trainable params: 0
```

```
#For early stopping
# Stop training when a monitored metric has stopped improving.
# monitor: Quantity to be monitored.
# patience: Number of epochs with no improvement after which training will be stopped.
import tensorflow as tf
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
# We use the “adam” optimizer, an algorithm that changes the weights and biases during training.
# We also choose binary-crossentropy as loss (because we deal with binary classification) and accuracy as our evaluation metric.
model.compile(
optimizer = "adam",
loss = "binary_crossentropy",
metrics = ["accuracy"]
)
from sklearn.model_selection import train_test_split
results = model.fit(
X_train, y_train,
epochs= 2,
batch_size = 500,
validation_data = (X_test, y_test),
callbacks=[callback]
)
# Let's check mean accuracy of our model
print(np.mean(results.history["val_accuracy"]))
# Evaluate the model
score = model.evaluate(X_test, y_test, batch_size=500)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
20/20 [==============================] - 1s 24ms/step - loss: 0.2511 - accuracy: 0.8986
Test loss: 0.25108325481414795
Test accuracy: 0.8985999822616577
```

```
#Let's plot training history of our model.
# list all data in history
print(results.history.keys())
# summarize history for accuracy
```

```
plt.plot(results.history['accuracy'])
plt.plot(results.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(results.history['loss'])
plt.plot(results.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```