

# DL lab assignment 2

February 10, 2025

```
[1]: # Import required libraries
import numpy as np
import pandas as pd
import seaborn as sns
from keras.datasets import imdb
from sklearn.model_selection import train_test_split
from keras import models
from keras import layers
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
[2]: # Load IMDB dataset and keep top 10,000 most frequent words
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
```

```
# Consolidate data for EDA (Exploratory Data Analysis)
data = np.concatenate((X_train, X_test), axis=0)
label = np.concatenate((y_train, y_test), axis=0)

# Checking shape of the dataset
print("X_train shape:", X_train.shape) # (25000,)
print("X_test shape:", X_test.shape) # (25000,)
print("y_train shape:", y_train.shape) # (25000,)
print("y_test shape:", y_test.shape) # (25000,)
```

```
X_train shape: (25000,)
X_test shape: (25000,)
y_train shape: (25000,)
y_test shape: (25000,)
```

```
[ ]: # Print first review
print("Review is:", X_train[0])

# Retrieve the vocabulary mapping from word to index
vocab = imdb.get_word_index()
print(vocab)

# Checking labels (0 = negative, 1 = positive)
print("Label of first review:", y_train[0]) # 1 (positive)
```

Review is: [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5 952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]

```
{"fawn": 34701, "tsukino": 52006, "nunnery": 52007, "sonja": 16816, "vani": 63951, "woods": 1408, "spiders": 16115, "hanging": 2345, "woody": 2289, "t rawling": 52008, "hold's": 52009, "comically": 11307, "localized": 40830, "disobeying": 30568, "royale": 52010, "harpo's": 40831, "canet": 52011, "ai leen": 19313, "accurately": 52012, "diplomat's": 52013, "rickman": 25242, "arranged": 6746, "rumbustious": 52014, "familiarness": 52015, "spider)": 520 16, "hahahah": 68804, "wood": 52017, "transvestism": 40833, "hangin": 34702, "bringing": 2338, "seamier": 40834, "wooded": 34703, "bravora": 52018, "grueling": 16817, "wooden": 1636, "wednesday": 16818, "prix": 52019, "altagracia": 34704, "circuitry": 52020, "crotch": 11585, "busybody": 57766, "t art'n'tangy": 52021, "burgade": 14129, "thrace": 52023, "tom's": 11038, "smuggles": 52025, "francesco": 29114, "complainers": 52027, "templarios": 521 25, "272": 40835, "273": 52028, "zaniacs": 52130, "275": 34706, "consenting": 27631, "snuggled": 40836, "inanimate": 15492, "uality": 52030, "bronte": 11926, "errors": 4010, "dialogs": 3230, "yomada's": 52031, "madman's": 34707, "dialoge": 30585, "usenet": 52033, "videodrome": 40837, "kid)": 26338, "pawed": 52034, "'girlfriend)": 30569, "'pleasure": 52035, "'reloaded)": 52036, "kazakos)": 40839, "rocque": 52037, "mailings": 52038, "brainwashed": 11927, "mcanally": 16819, "tom)": 52039, "kurupt": 25243, "affiliated": 21905, "babaganoosh": 52040, "noe's": 40840, "quart": 40841, "kids": 359, "up lifting": 5034, "controversy": 7093, "kida": 21906, "kidd": 23379, "error": 52041, "neurologist": 52042, "spotty": 18510, "cobblers": 30570, "project
```

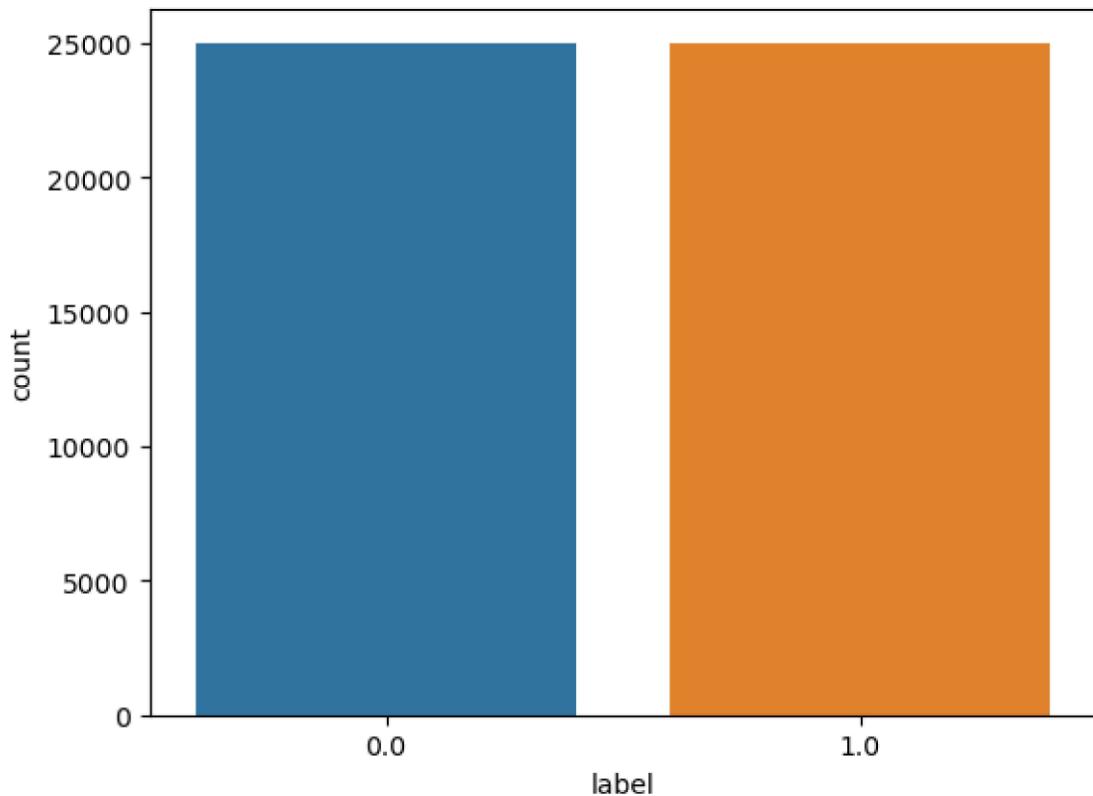
```
[4]: # Function to vectorize the sequences (binary matrix representation)
def vectorize(sequences, dimension=10000):
    # Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))

    # Set appropriate positions to 1 based on the sequence
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1
    return results
```

```
[5]: # Apply vectorization to data
data = vectorize(data)
label = np.array(label).astype("float32")

# Create DataFrame for label distribution
labelDF = pd.DataFrame({'label': label})
sns.countplot(x='label', data=labelDF)
```

```
[5]: <Axes: xlabel='label', ylabel='count'>
```



```
[6]: # Create training and testing datasets
train_x = data[10000:]
train_y = label[10000:]
test_x = data[:10000]
test_y = label[:10000:]

# Check the shape of the training and testing sets
print("train_x shape:", train_x.shape) # (40000, 10000)
print("test_x shape:", test_x.shape) # (10000, 10000)
```

```
train_x shape: (40000, 10000)
test_x shape: (10000, 10000)
```

```
[7]: # Check the number of unique words in the dataset
print("Number of unique words:", len(np.unique(np.hstack(data))))
```

```
# Calculate average review length and its standard deviation
length = [len(i) for i in data]
print("Average Review length:", np.mean(length))
print("Standard Deviation:", round(np.std(length)))
```

```
Number of unique words: 2
Average Review length: 10000.0
Standard Deviation: 0
```

```
[ ] :
```

```
# Decode the first review to human-readable form
reverse_index = dict([(value, key) for (key, value) in vocab.items()])
decoded = " ".join([reverse_index.get(i - 3, "#") for i in data[0]])
print("Decoded Review:", decoded)
```

```
[9]: # Split the data into training and testing sets using train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, label, test_size=0.
                                                    ↪20, random_state=1)
print("X_train shape:", X_train.shape)    # (40000, 10000)
print("X_test shape:", X_test.shape)      # (10000, 10000)
```

```
X_train shape: (40000, 10000)
```

```
[10]: # Build a sequential model for DNN
model = models.Sequential()

# Input layer (Dense layer with 50 neurons and ReLU activation)
model.add(layers.Dense(50, activation="relu", input_shape=(10000,)))

# Hidden layers with dropout for regularization
model.add(layers.Dropout(0.3))
model.add(layers.Dense(50, activation="relu"))
model.add(layers.Dropout(0.2))
```

```

model.add(layers.Dense(50, activation="relu"))

# Output layer with sigmoid activation (for binary classification)
model.add(layers.Dense(1, activation="sigmoid"))

# Summarize the model architecture
model.summary()

```

C:\Users\SOFT LAB17\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.  
super().\_\_init\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)

Model: "sequential"

Layer (type)	Output Shape	
Param #		
dense (Dense)	(None, 50)	
↳ 500,050		
dropout (Dropout)	(None, 50)	
↳ 0		
dense_1 (Dense)	(None, 50)	
↳ 2,550		
dropout_1 (Dropout)	(None, 50)	
↳ 0		
dense_2 (Dense)	(None, 50)	
↳ 2,550		
dense_3 (Dense)	(None, 1)	
↳ 51		

Total params: 505,201 (1.93 MB)

Trainable params: 505,201 (1.93 MB)

Non-trainable params: 0 (0.00 B)

```
[11]: # Early stopping callback to prevent overfitting
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
```

```
[12]: # Compile the model with Adam optimizer and binary crossentropy loss
model.compile(optimizer="adam", loss="binary_crossentropy",  
               metrics=["accuracy"])
```

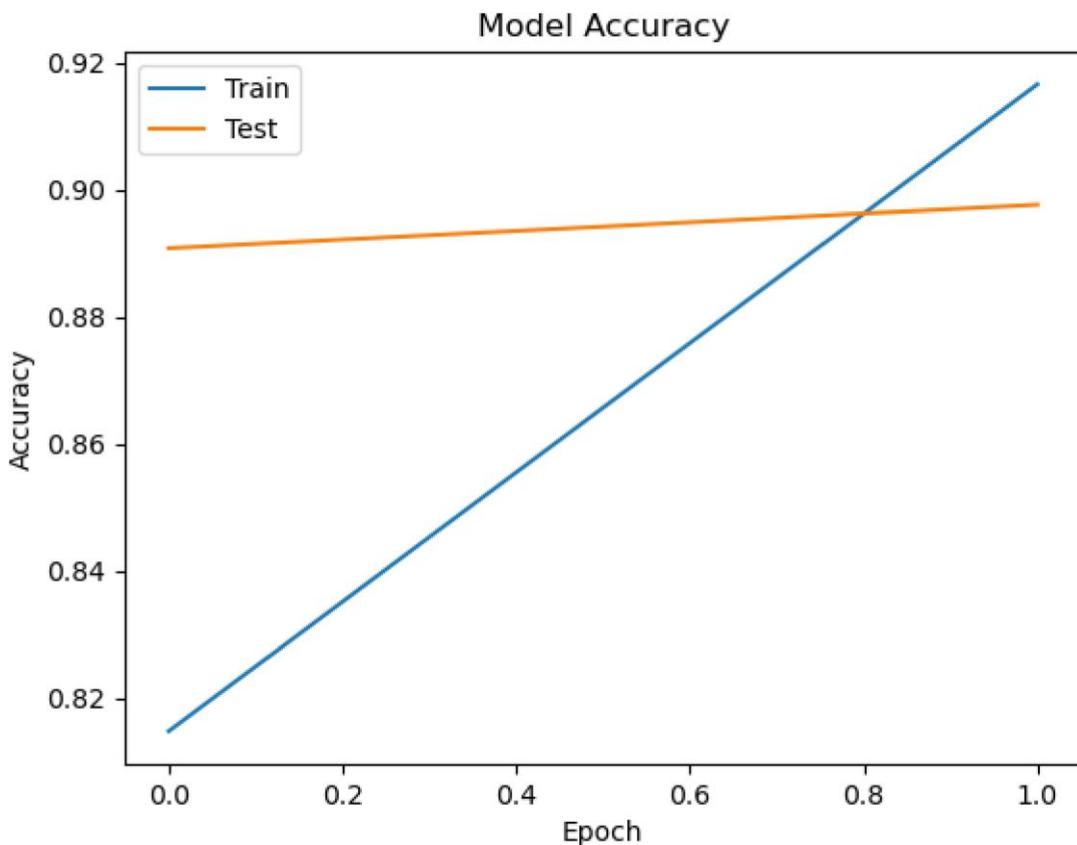
```
[13]: # Train the model
results = model.fit(X_train, y_train, epochs=2, batch_size=500,  
                     validation_data=(X_test, y_test), callbacks=[callback])
```

```
Epoch 1/2
80/80          5s 37ms/step -
accuracy: 0.7220 - loss: 0.5366 - val_accuracy: 0.8908 - val_loss: 0.2653
Epoch 2/2
80/80          2s 20ms/step -
accuracy: 0.9161 - loss: 0.2176 - val_accuracy: 0.8976 - val_loss: 0.2567
```

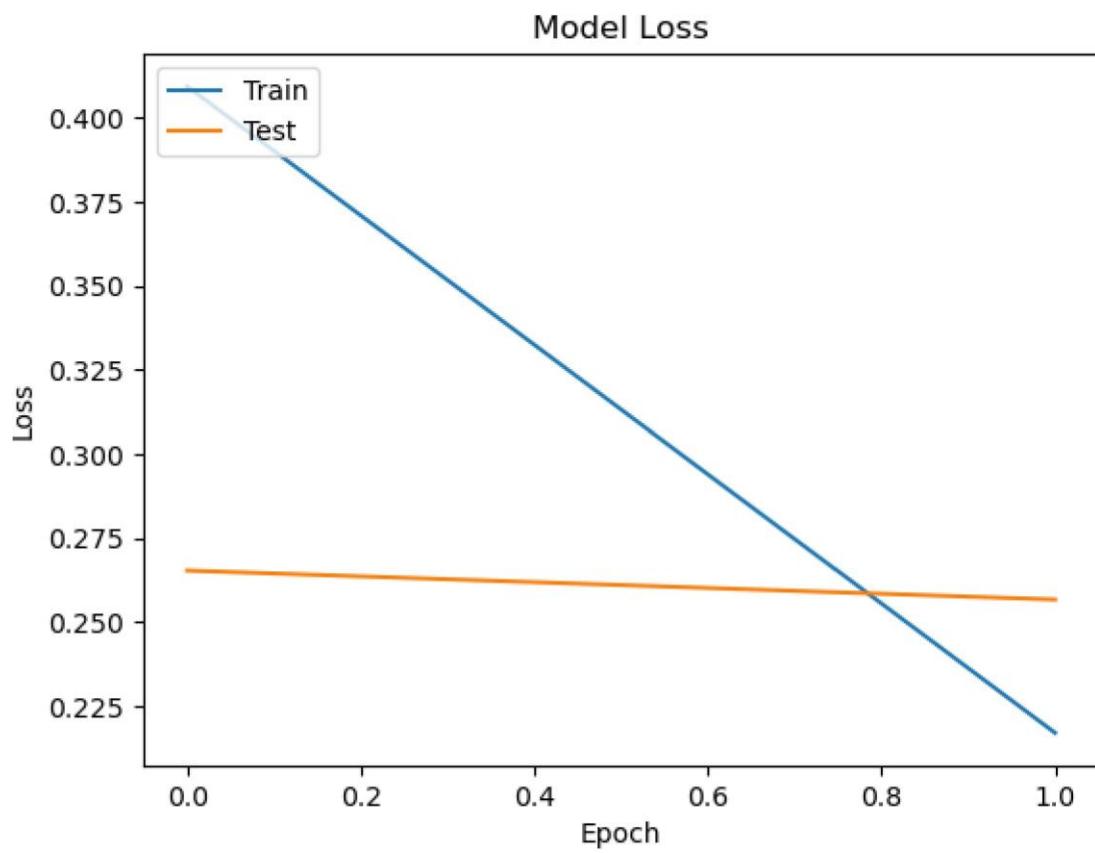
```
[14]: # Evaluate the model on test data
score = model.evaluate(X_test, y_test, batch_size=500)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
20/20          0s 10ms/step -
accuracy: 0.8937 - loss: 0.2619
Test loss: 0.25669997930526733
Test accuracy: 0.897599995136261
```

```
[15]: # Plot training and validation 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()
```



```
[16]: # Plot training and validation 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()
```



[ ] : Conclusion- In this way we can Classify the Movie Reviews by using DNN.