## EXP-5 (Build Logistic Regression Classifier using Neural Networks)

by Allena Venkata Sai Abhishek (122021601009)

```
import tensorflow_datasets as tfds
import tensorflow as tf
import numpy as np
```

```
#importing the data in to X & y
#since this is a very small dataset only train set is present
X, y  = tfds.as_numpy(tfds.load('iris',split='train',batch_size=-1,as_supervised=True,))
```

```
Downloading and preparing dataset iris/2.0.0 (download: 4.44 KiB, generated: Unknown
Dl Completed...: 100%  1/1 [00:00<00:00, 2.26 url/s]
Dl Size...:  0/0 [00:00<?, ? MiB/s]

Shuffling and writing examples to /root/tensorflow_datasets/iris/2.0.0.incomplete1Z0H
0%  0/150 [00:00<?, ? examples/s]
Dataset iris downloaded and prepared to /root/tensorflow_datasets/iris/2.0.0. Subsequ
```

```
#printing the top values of the feature variable
X
```

```
[6.2, 2.9, 4.3, 1.3],
[4.4, 2.9, 1.4, 0.2],
[5.7, 2.8, 4.1, 1.3],
[5.7, 2.5, 5. , 2. ],
[5.6, 3. , 4.5, 1.5],
[5.1, 3.3, 1.7, 0.5],
[5.5, 2.3, 4. , 1.3],
[6.3, 3.3, 4.7, 1.6],
[6.3, 2.8, 5.1, 1.5],
[6.3, 3.4, 5.6, 2.4],
[6.5, 3. , 5.5, 1.8],
[4.6, 3.1, 1.5, 0.2],
[7.7, 3.8, 6.7, 2.2],
[5. , 2. , 3.5, 1. ],
[7.2, 3.6, 6.1, 2.5],
[6.4, 3.2, 4.5, 1.5],
[4.9, 3. , 1.4, 0.2],
[5.9, 3. , 4.2, 1.5],
[4.8, 3.1, 1.6, 0.2],
[6.3, 3.3, 6. , 2.5],
[6.4, 2.8, 5.6, 2.2],
[4.8, 3.4, 1.9, 0.2],
[6.7, 2.5, 5.8, 1.8],
[5.1, 3.8, 1.9, 0.4],
[5.7, 2.9, 4.2, 1.3],
[6.4, 2.7, 5.3, 1.9],
[5.4, 3.4, 1.7, 0.2],
```

```
[6.8, 3. , 5.5, 2.1],
[6.3, 2.5, 4.9, 1.5],
[4.4, 3.2, 1.3, 0.2],
[4.9, 3.1, 1.5, 0.1],
[7.7, 2.6, 6.9, 2.3],
[5. , 3.4, 1.5, 0.2],
[6. , 2.7, 5.1, 1.6],
[6.7, 3.3, 5.7, 2.5],
[5. , 3.5, 1.6, 0.6],
[6.4, 3.2, 5.3, 2.3],
[7.4, 2.8, 6.1, 1.9],
[6.9, 3.1, 5.4, 2.1],
[5.9, 3.2, 4.8, 1.8],
[5.5, 2.5, 4. , 1.3],
[6.2, 2.2, 4.5, 1.5],
[4.4, 3. , 1.3, 0.2],
[5. , 2.3, 3.3, 1. ],
[6.6, 3. , 4.4, 1.4],
[6.1, 2.9, 4.7, 1.4],
[6.2, 2.8, 4.8, 1.8],
[4.8, 3.4, 1.6, 0.2],
[5.9, 3. , 5.1, 1.8],
[6. , 3.4, 4.5, 1.6],
[7.2, 3.2, 6. , 1.8],
[5.8, 2.6, 4. , 1.2],
[4.7, 3.2, 1.3, 0.2],
[6.2, 3.4, 5.4, 2.3],
[5.1, 3.8, 1.6, 0.2],
[4.9, 2.4, 3.3, 1. ],
[6.7, 3.1, 5.6, 2.4],

[5.5, 2.4, 3.8, 1.1],
[4.9, 3.1, 1.5, 0.1]], dtype=float32)
```

```
#finding unique labels in the class variable
np.unique(y)
```

```
array([0, 1, 2])
```

# Preprocessing

this is a multi class dataseet but we are trying to convert it into a binary class

0 (Iris-setosa)

1 (Iris-versicolor)

2 (Iris-virginica)

here we are converting the dataset into

0 (Iris-setosa)

1 (not a setosa)

keeping only sentosa and changing the rest into not a sentosa

```
# converting multi-class objhects to a binary class
def convert_binary(y):
  new_y = np.zeros(len(y),dtype=np.int8)
```

```
  for i in range(len(y)):
    if(y[i] != 0):
      new_y[i] = 1
  return new_y


binary_y = convert_binary(y)
np.unique(binary_y)
```

```
array([0, 1], dtype=int8)
```

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

layer = model.layers[0]
print("Initial Weights")
print("Layer:",layer)
print("layer weights:  ",layer.weights)
print("layer bias initializer:  ",layer.bias_initializer)
```

```
Initial Weights
Layer: <tensorflow.python.keras.layers.core.Dense object at 0x7f5101918150>
layer weights:   []
layer bias initializer:   <tensorflow.python.keras.initializers.initializers_v2.Zeros
```

```
history = model.fit(X,binary_y, steps_per_epoch=10, epochs=100, verbose=1, validation_spli
```

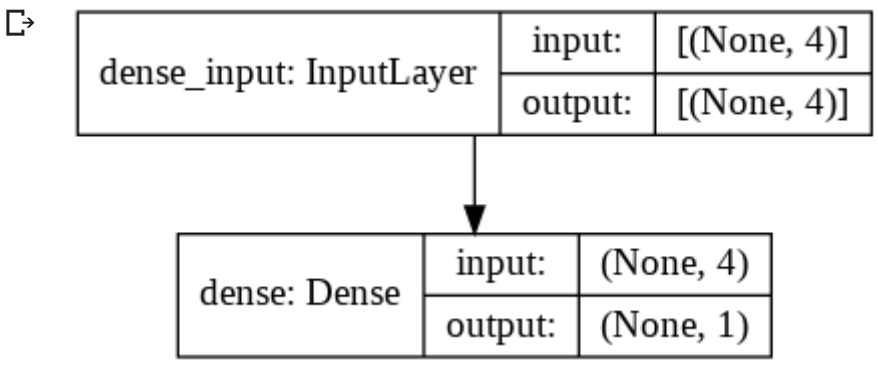
```
Epoch 72/100
10/10 [==============================] - 0s 4ms/step - loss: 0.2198 - accuracy: 0.9
Epoch 73/100
10/10 [==============================] - 0s 4ms/step - loss: 0.2175 - accuracy: 0.9
Epoch 74/100
10/10 [==============================] - 0s 4ms/step - loss: 0.2151 - accuracy: 0.9
Epoch 75/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2129 - accuracy: 0.9
Epoch 76/100
10/10 [==============================] - 0s 4ms/step - loss: 0.2106 - accuracy: 1.0
Epoch 77/100
10/10 [==============================] - 0s 4ms/step - loss: 0.2083 - accuracy: 1.0
Epoch 78/100
10/10 [==============================] - 0s 4ms/step - loss: 0.2062 - accuracy: 1.0
Epoch 79/100
10/10 [==============================] - 0s 4ms/step - loss: 0.2041 - accuracy: 1.0
Epoch 80/100
10/10 [==============================] - 0s 4ms/step - loss: 0.2019 - accuracy: 1.0
Epoch 81/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1998 - accuracy: 1.0
Epoch 82/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1977 - accuracy: 1.0
Epoch 83/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1957 - accuracy: 1.0
Epoch 84/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1937 - accuracy: 1.0
Epoch 85/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1917 - accuracy: 1.0
```

```
Epoch 86/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1898 - accuracy: 1.0
Epoch 87/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1879 - accuracy: 1.0
Epoch 88/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1861 - accuracy: 1.0
Epoch 89/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1840 - accuracy: 1.0
Epoch 90/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1824 - accuracy: 1.0
Epoch 91/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1804 - accuracy: 1.0
Epoch 92/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1786 - accuracy: 1.0
Epoch 93/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1769 - accuracy: 1.0
Epoch 94/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1751 - accuracy: 1.0
Epoch 95/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1734 - accuracy: 1.0
Epoch 96/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1718 - accuracy: 1.0
Epoch 97/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1701 - accuracy: 1.0
Epoch 98/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1685 - accuracy: 1.0
Epoch 99/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1668 - accuracy: 1.0
Epoch 100/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1652 - accuracy: 1.0
```

```
score, accuracy = model.evaluate(X[1:75], binary_y[1:75], batch_size=16, verbose=0)
print("NN-Score = {:.2f}".format(score))
print("NN-Accuracy = {:.2f}".format(accuracy))
```

```
NN-Score = 0.17
NN-Accuracy = 1.00
```

```
#plotting the model of logistic regression here is is in a simple preceptron style
tf.keras.utils.plot_model(model, show_shapes=True)
```




```
#Confusion matrix
y_pred = model.predict(X)
```

```
tf.math.confusion_matrix(binary_y, y_pred )
```

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[ 50,   0],
       [100,   0]], dtype=int32)>
```

```
#Plotting metric curves
import matplotlib.pyplot as plt
```

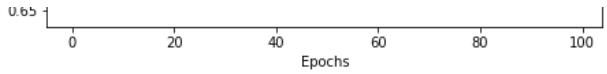
```
plt.figure(figsize=(16, 8))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel("Epochs")
plt.ylabel('accuracy')
plt.legend(['accuracy', 'val_accuracy'])
plt.ylim(None, 1)

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel("Epochs")
plt.ylabel('loss')
plt.legend(['loss', 'val_loss'])
plt.ylim(0, None)
```

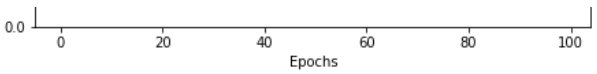
```
(0.0, 1.3760351985692978)
```



```
layer = model.layers[0]
print("Layer:",layer)
print("layer weights:  ",layer.weights)
print("layer bias:  ",layer.bias.numpy())
print("layer bias initializer:  ",layer.bias_initializer)
```

```
Layer: <tensorflow.python.keras.layers.core.Dense object at 0x7f5101918150>
layer weights:   [<tf.Variable 'dense/kernel:0' shape=(4, 1) dtype=float32, numpy=
array([[ 0.28385147],
       [-0.8314129 ],
       [ 0.52591366],
       [ 0.7323479 ]], dtype=float32)>, <tf.Variable 'dense/bias:0' shape=(1,) dtype=
layer bias:   [-0.5067841]
layer bias initializer:   <tensorflow.python.keras.initializers.initializers_v2.Zeros
```




✓ 0s completed at 9:20 AM